The application of parallel processing to diesel engine modelling

Jones, A. D.

Award date:
1987

Awarding institution:
University of Bath

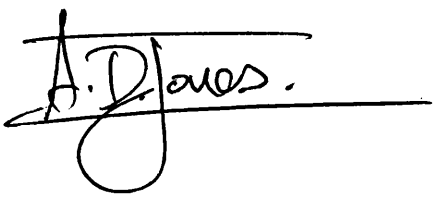
The Application of Parallel Processing to Diesel Engine Modelling

submitted by A. D. Jones

for the degree of Ph.D.

at the University of Bath

1987



ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

SUMMARY

The programme of research which is reported in this thesis describes a new approach to improving the speed of simulation of diesel engines. The engine is represented by a filling and emptying model which is divided into a number of modules; these modules are then calculated in parallel, and with a significant improvement in speed. Division of the model was made along control volume boundaries, and a single processing node used to compute each volume (or a number of volumes). A tightly coupled parallel computer system was developed to compute the model using MC68000 and MC68020 based processing nodes, and a modified version of the TRIPOS operating system to oversee its operation.

A diesel engine simulator was developed based on this solution technique which incorporates other novel features, such as an animated display of engine model performance.

Experiments were carried out to measure how much faster the engine model is computed in parallel compared to serial computation, and results show that an improvement as high as ten is possible. Experiments have also been performed to measure the sensitivity of the execution speed to other factors, such as the use of look up tables and integration step size.

Finally, a number of recommendations have been made, which if implemented, will further increase the execution speed of the model and improve the usefulness of the simulator.

ACKNOWLEDGEMENTS

I would like to express my gratitude to Mr A. R. Daniels, Dr N. Vaughan and Dr S. J. Charlton for their help and encouragement in supervising this work. I should also like to express by appreciation for the help given by many colleagues in the Department, with special thanks to Dr L. Dale, C. Selwyn, R. Dunn, Dr S. Williams and T. Berry.

I would like to acknowledge the assistance of members of the Department of Mechanical Engineering in this work, in particular to Professor F. Wallace for providing access to the engine test bed facilities.

I should like to thank my fiancee, for typing this thesis and also my father for checking the text and providing many helpful suggestions.

Finally, I wish to thank Professor F. Eastham for permission to study in the School of Electrical Engineering and the Science and Engineering Research Council for providing a grant.

CONTENTS

CHAPTER 5

CHAPTER 6

CHAPTER 7

CHAPTER 8

CHAPTER 9

CHAPTER 10

CHAPTER 11

NOTATION

List of Principle Symbols

A		area	m^2
C		thermal capacitance	J/K
	also	speed of sound	m/s
C_d		discharge coefficient	-
C_r		compression ratio	-
crl		connecting rod length	m
d		diameter (of cylinder)	m
f		fuel/air ratio	-
h		heat transfer coefficient	W/m^2K
ho		specific stagnation enthalpy	J/Kg
id		ignition delay	ms
J		inertia	Kgm^2
k		gain	-
m		mass	Kg
P		pressure	N/m^2
	also	power	W
Q		heat transfer	J
R		gas constant	J/KgK
	also	thermal resistance	K/W
r		radius (crank throw)	m
T		temperature	K
	also	torque	Nm
t		time	s
u		specific internal energy	J/Kg
V		volume	m^3
$\bar{V}_p$		mean piston speed	m/s
x		actuator position	m
z		fuel rack position	m
β		premixed fuel fraction	-
$\Delta\theta$		integration step size	radians
γ		ratio of specific heats	-
ζ		damping ratio	-
η		efficiency	%
θ		crank angle (from TDC)	radians
τ		time (non dimensional)	-
ω		angular velocity	radians/s
ω_n		undamped natural frequency	radians/s

Suffices

c		compressor
d		diffusion controlled (combustion)
	also	downstream conditions
e		engine
ev		exhaust valve
f		fuel
fb		fuel burnt
for		formation

i	in (flow)
iv	inlet valve
o	out (flow)
p	pre-mixed controlled (combustion)
s	stoichiometric
t	turbine
tc	turbocharger
u	upstream conditions
w	wall

CHAPTER 1

1.1 Introduction

It is well known that progress in most areas of technology has usually been the result of economic pressures to do, or to make things more cheaply. The development of engines is a classic example of this. The diesel engine, for example, was developed because of Rudolph Diesel's drive and determination to design an engine which would operate more efficiently and be cheaper to run than other power units then in use.

There are, of course, other factors apart from economics, which stimulate technology. The twentieth century has seen how advanced military requirements and their associated research programmes result in technological progress. Improvements in technology can also be brought about by adopting, or adapting, contemporary ideas and new techniques from another technology, and often quite dramatic improvements can be brought about in this way. The research which is the subject of this thesis is an example of this, where worthwhile progress has been made in the technique of diesel engine simulation by applying the latest parallel computing techniques, and as far as is known, is the first time that this has been done.

Before discussing the research in detail it is interesting and instructive to review briefly the history of diesel engine development, so that recent progress can be seen in proper

perspective.

1.2 Historical Development

The first diesel engine to operate successfully was run by Rudolph Diesel in 1897: it had only one cylinder which operated on a four stroke power cycle and developed a maximum power of 13 kW at 154 rpm [1.1]. It is interesting to note that it had a power to weight ratio of 275 kg per kW, compared with 3 kg per kW for a modern high speed turbocharged engine.

The early engines were so large and heavy that they could only be used in stationary applications. Considerable effort was therefore devoted to improving the power developed from a given size of engine, and by the First World War the diesel engine had become the standard power plant for submarines, where it proved to be very successful.

Since power output is determined mainly by the quantity of fuel which the engine can burn per cycle, one obvious way to increase power was to make the engine run faster, and indeed this was the stimulus for the development of the "high" speed diesel engine in the 1920's. These engines were light enough to be considered for use in transport and in the 1930's a whole range of applications quickly followed such as use in trains and trucks, and throughout the Second World War the diesel engine was the primary power unit on land and sea.

Further improvements in engine performance were made by

increasing the quantity of air supplied to the engine in each power stroke so that a corresponding increase could be made in the quantity of fuel injected into the cylinders - and hence the engine could generate more power. Since air density increases with pressure, the mass of air taken into the engine can be increased by pressurizing the air before it enters into the combustion chamber. Both the mechanical-supercharger and the turbo-supercharger were used to do this, although the relatively poor specific fuel consumption of the mechanical-supercharger has led to its demise. The first experiments using turbochargers were carried out by Buchi [1.2], who patented the first type of turbocharger in 1915. Several more years elapsed before the first successful system was developed and applications were almost entirely restricted to low speed engines until the beginning of the Second World War. The use of the turbocharger has greatly improved the competitive position of the diesel engine because its size, weight and running costs are all significantly less than those of a naturally aspirated engine of similar power.

The success of the diesel has been due largely to its reliability and economical use of fuel. This is especially so in those applications requiring more or less continuous operation, where economy in use of fuel can more than offset the disadvantage of high initial capital cost. For example, in maritime applications fuel cost is the most significant running cost to be considered in the through life costing of a ship [1.3]. However, the diesel has not yet been able to achieve the same dominant position as a power unit for cars, because the higher initial cost still outweighs the

reduced running cost, in all but high mileage applications. Even so, the high cost of fuel in recent years has improved the competitiveness of the diesel and most popular makes of car are now offered with a diesel engine option.

1.3 Future Trends

Although the diesel engine has now reached a very advanced state of development with good fuel consumption and reliability, diesel engine manufacturers and research institutions continue to invest heavily in research and development in order to ensure its future. This includes the ever continuing evolutionary process of making improvements by paying greater attention to conventional engine design, and more recently, work aimed at developing radical innovations. Examples of major innovation include work aimed at totally eliminating heat losses from an engine and operation of the engine without a lubricating oil. These radical approaches to engine design are inevitably associated with high technical and financial development risks, but if successful, should result in major improvements to engine performance. Reviews of future engine developments are given in references 1.2, 1.4-1.7.

Another approach to improving engine performance, which has been the subject of much recent research is to exercise improved control, either directly or indirectly, over those critical parameters which determine how an engine responds - essentially engine fuel and air supplies. Research aimed at developing variable geometry turbochargers [1.2,1.8], variable valve timing [1.9],

improved fuel injection control [1.10] and so on, all belong to this category. A particular example of this kind of research is that being carried out on the differential compound engine, under development by Wallace [1.11]. Admittedly, this is a combined engine and transmission system, nevertheless control is being exercised over as many as five separate inputs in order to provide a high degree of control over the response of the total system. The five inputs are fuel rack position control, fuel injection timing control, bypass valve control, turbine nozzle control and variable ratio gearbox control, as is shown in the schematic of the engine given in Figure 1.1. Thus as further opportunities are presented to control those critical parameters which determine engine response, so engine control becomes more sophisticated, but this sophistication is essential if the maximum benefits are to be obtained, such as reduced fuel consumption, low emissions and good transient performance. Before discussing in greater detail how this sophisticated control can be applied, it is necessary to first consider the type of response which is required from a diesel engine.

The response required depends very much on the application. In transportation the primary requirement is to give the vehicle what is generally referred to as good "driveability" with, of course, good fuel economy and adequately low exhaust pollution. Driveability is a rather subjective representation of the "feel" which a driver obtains from the performance of the vehicle, depending on how smoothly and briskly the vehicle responds to changes in required speed. With electrical power generation, the

primary requirement is to maintain tight control of the engine speed whatever the load; again good fuel economy is desirable and the engine must not exceed permitted levels of pollution. Not only can different applications present different requirements, but these requirements can also change quite significantly in any one application in a very short period of time: for example, when a truck is overtaking another vehicle the performance required (ie good acceleration) is quite different to that needed when the truck is just cruising along.

It is evident that the engine controller must be capable of providing a very versatile performance, and when it is further recognised that transient performance, fuel economy and emissions can be traded off, to some extent one against another, the degree of versatility to be provided starts to become more clearly seen. The controller must be designed to control in an "intelligent" manner all the inputs to the engine so as to give good performance whatever the immediate requirement is, ie good dynamic response during speed transients, and at all times to minimise the quantity of fuel burnt, without exceeding emissions levels. This type of problem, where a number of variables are controlled in such a way as to maximise some pre-determined measure of overall performance, is an obvious application for optimum control methods. There are, however, some serious practical problems to be overcome before these very sophisticated controllers can be used. This is because the solution of the optimisation equation requires an immense amount of computation and even the very powerful microprocessors available today are unable to solve the relevant equations quickly enough to

be generally useful.

Because of these practical difficulties, all sophisticated engine controllers used so far employ an open loop scheduled type of control. The structure of such a controller is shown in Figure 1.2. At the lowest level are the actuators which physically set the inputs to the engine, and the transducers for measuring the resulting responses. The demand signals for the actuators are supplied by the hierarchical stage which is a controller based on a microprocessor. The hierarchical stage uses measurements of the engine parameters such as temperature, speed, boost pressure and fueling to "calculate" the required control action. Because of the computational difficulties outlined above, it is usual for the controller to select the "best" settings for the engine inputs using a procedure based upon the actual measurements made and appropriate entries into a pre-determined "look up" table. The "look up" tables are normally based on the nominal steady state performance of that engine [1.12], and the inevitable result is a less than optimum performance from the controller. For example, such a controller cannot compensate for differences in performance between one engine and another, or for the effect of wear during the course of engine life, or for changes in ambient conditions, fuel quality, and so on.

1.4 Engine Simulation

When developing engine controllers, there are many advantages in simulating the behaviour of the engine rather than using a real one. Simulation has the important advantage that it can be used

quickly to explore the benefits of different control strategies, without requiring changes to be made to engine hardware, and without the possibility of subjecting the engine to control inputs which may be dangerous.

For the simulation to provide a realistic representation of the engine behaviour under both steady and dynamic conditions, it is necessary to represent the engine in considerable detail (such as by the filling and emptying type model which will be described in Chapter 4). Unfortunately, such detailed models are extremely demanding in computational effort and this can be the dominant factor in determining whether or not the model can be used in a controller design study [1.13]. For example, when using a filling and emptying model, long execution times, possibly of the order of hours, are required to compute the behaviour of a typical high speed engine over but a few seconds of real time.

In order to achieve the fastest execution speed, care must be taken to ensure that only the essential aspects of engine behaviour are modelled and that the numerical methods used to solve the equations are powerful and fast. Additionally the program must be carefully structured so that it will run as fast as possible. Having then reached this point, any further significant increase in speed would normally be sought by employing a faster processor unit, although this is not always possible, because of cost considerations.

An alternative approach for improving the computational speed of the engine model and which is the central topic of this research,

is to divide the engine model into a number of smaller units and to compute their solutions in parallel. Conventional computer systems presently used for engine simulation, perform one operation at a time using a single processor and their speed can only be increased by increasing the speed of their internal operations. On the other hand, parallel computers achieve their increased speed by performing more than one operation at a time. Using the powerful microprocessor devices which are now available, it is possible to design fast parallel computer systems which are relatively compact and inexpensive, and in certain applications it may be viable to dedicate the computer system to the application. Indeed, a highly desirable long term goal for this programme of work is to be able to solve the engine model equations in a time which is no greater than the time which it takes the real engine to perform the same task, ie in real time. Real time solution of the model equations opens up the possibility of using the model in systems which incorporate engine hardware. For instance, in engine control the model might be used as an observer to predict the values of engine variables which are needed to provide better control, but which are difficult, or expensive to measure. In condition monitoring the model could be used to oversee the operation of an engine, and to warn, or shut down the engine, if a hazardous operating condition were likely to occur. These are exciting new developments which are almost within reach, and which when implemented should result in further significant improvements in engine operation and control.

1.5 Contents

The contents of the thesis are arranged as follows. Chapter 2 provides a general background to the subject of engine simulation, including a review of the historical development of engine simulation and of the different approaches which have been used to model engines. Chapter 3 describes an experimental Leyland TL11 engine which has been simulated in order to assess the practical value of the research which has been undertaken. The engine model used in the simulation is described in Chapter 4, and the manner in which it has been subdivided in order to apply parallel processing is discussed in Chapter 5. The parallel processing computer system hardware and operating system software developed in order to carry out the research programme, is described in Chapter 6. Chapter 7 describes the software which was developed to implement the parallel processing solution chosen, and Chapter 8 shows results from the engine model which were recorded when testing the software. Results showing the improvement in execution speed which has been achieved, are given in Chapter 9. Finally, Chapter 10 presents the conclusions emerging from the research, and Chapter 11 makes recommendations for possible future work.

1.6 References

1.1 C L Cummins.
Internal fire.
1976, Carnot Press, Lake Oswego, Oregon.

1.2 N Watson.
Turbochargers for the 1980's - Current Trends and Future Prospects.
1979, SAE 790063.

1.3 R V Thompson.
Marine Technology - present and future.
1985, Proc. I.Mech.E Vol 199.

1.4 R W Dugan, J E Perry.
Diesel Engines - Designing for the Future.
1982, SAE 820621.

1.5 R Kamo, L Tozzi, R Sekar.
Light Duty Vehicle Diesel Engine Assessment Program - Passenger Car Diesel Engine of the Future.
1983, 20th Auto. Tech. Development Contractor's Coordination Meeting, P-120 by SAE.

1.6 L Williamson.
Truck Engines - The Future.
1982, Technology Seminar.

1.7 W Bryzik.
TACOM/Cummins Adiabatic Engine Program.
1983, SAE P-120.

1.8 F J Wallace, R J B Way, A Baghery.
Variable Geometry Turbocharging - The Realistic Way Forward.
1981, SAE 810336.

1.9 C R Stone, E K M Kwan.
Variable valve timing for ic engines.
1985, Automotive Engineer.

1.10 P E Glikin.
Fuel injection in diesel engines.
1985, Proc. I.Mech.E Vol 199.

1.11 F J Wallace, M Tarabad, D Howard.
The Differential Compound Engine - a new Integrated Engine Transmission System concept for heavy vehicles.
1983, Proc. I.Mech.E, Vol 197A.

1.12 L M Sweet.
Control Systems for Automotive Vehicle Economy: A Literature Review.
1981, J. Dynamic Systems, Measurement, and Control.

1.13 N Watson.
Dynamic Turbocharged Diesel Engine Simulator for Electronic Control System Development.
1984, J. of Dynamic Systems, Measurement and Control, Vol 106.

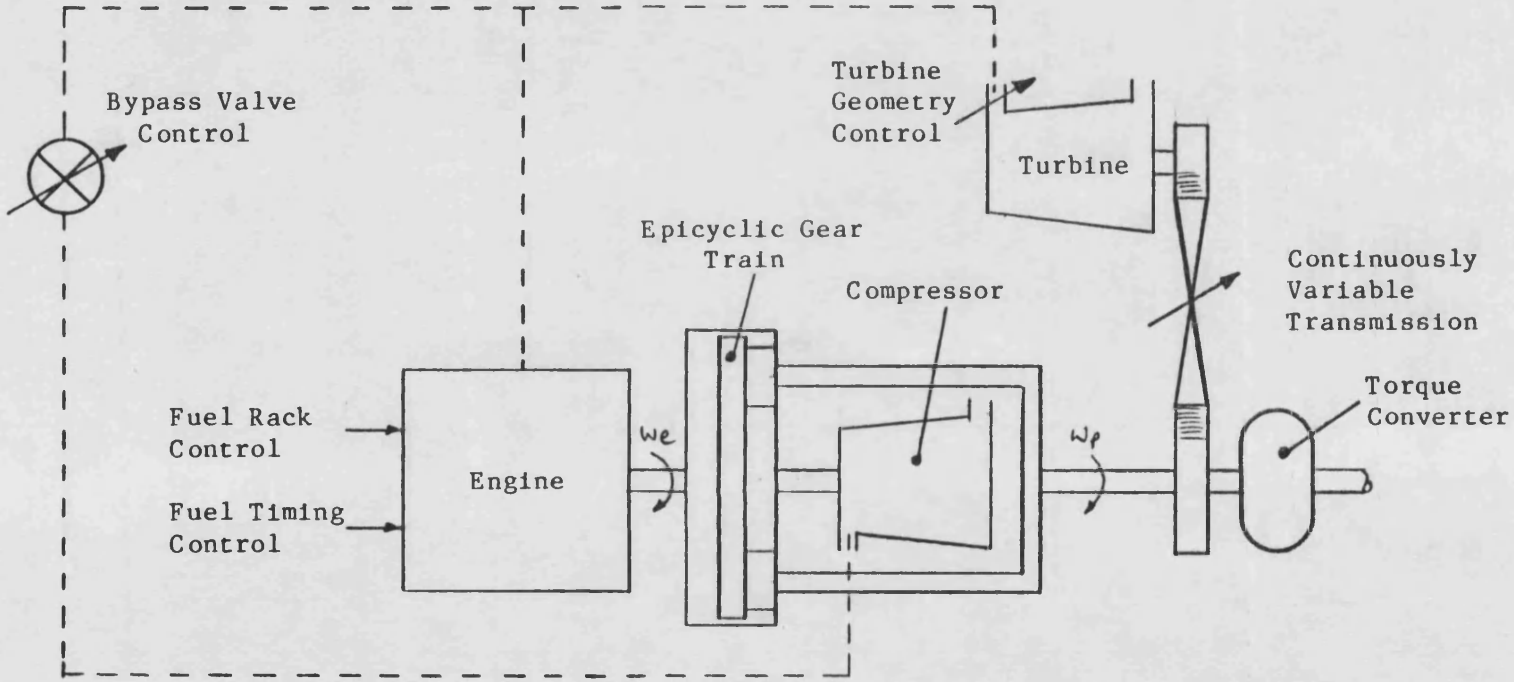


Figure 1.1 Differential Compound Engine Layout

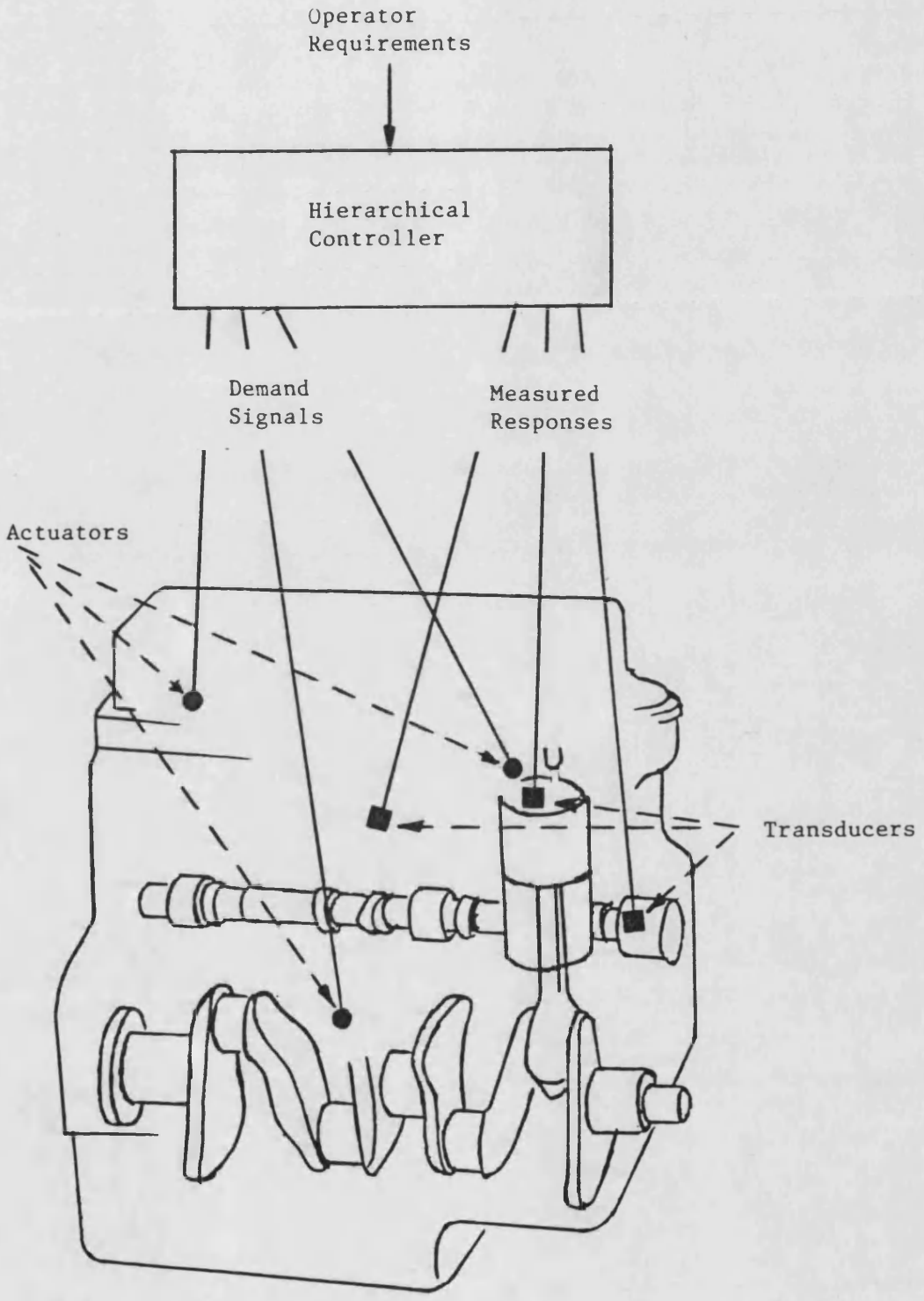


Figure 1.2 Schematic of a Sophisticated Engine Controller

CHAPTER 2

2.1 Engine Simulation

Simulation of engine performance is widely used by control engineers and engine designers to assist them in their tasks, and even a cursory study of the literature shows that virtually every approach to engine modelling seems to have been tried, at some time or other. This reflects the diversity of tasks for which the models are used and the many difficulties which have been encountered in obtaining a satisfactory representation of engine behaviour.

A brief review is now given of the historical development of dynamic engine models, followed by a discussion of different approaches to engine modelling.

2.2 Historical Development of Dynamic Engine Models

The development of dynamic engine models has been inextricably linked with the activities of the engineers of the time, and the problems they were attempting to overcome. The first problems of any serious practical importance arose in the control of steam engines. Early progress in steam engine control had been painfully achieved over a long period of time and without any significant theoretical understanding emerging. As steam engine design improved and manufacturing methods also gradually improved, instability problems began to be encountered, such as speed hunting, and by the

middle of the 19th century these problems had become so serious that they attracted the attention of theoretical scientists. Clerk Maxwell [2.1] studied the equation of motion of a closed loop system and showed that the behaviour of a feedback mechanism can be described by a linear differential equation and that the stability of the feedback loop can be analysed from a consideration of the roots of the equation. Routh [2.2] extended this work and also discussed further the stability of such systems. Unfortunately, these analytic methods rarely provided the correct solution when applied to engine analysis and understandably were neglected by the more practical engine manufacturers [2.3]. This lack of success stultified further theoretical study and regrettably the design of engine controllers quickly reverted to an art once again.

Renewed attempts to analyse the dynamic behaviour of diesel engines were made in the 1940's to 1960's, using continuous control methods [2.3-2.6]. The process whereby energy is converted into power was represented by a simple time delay as though fuel is injected continuously into the engine with each element of fuel producing its contribution to the torque after an appropriate delay. Clearly this is a simplification, since the engine actually only receives injections of fuel at discrete instants in time - and the engine is essentially uncontrolled between these injections. Attempts to improve this inadequacy led to the application of sampled data theory [2.7-2.9], to represent the injection of fuel as a series of discrete events. This in turn, resulted in considerable interest being shown in the use of system identification techniques to obtain improved linear models representing diesel engine

behaviour. A variety of test signals were used to perturb the engine in the system identification studies, including sinewaves, steps [2.10,2.11], and pseudo random binary sequences [2.12,2.13]. Although several theoretical controllers were designed using the results of the experiments, few appear to have been implemented practically.

The early 1970's saw diesel engine manufacturers once again experiencing considerable problems with the speed control of engines; this time the problems occurred on engines which had been turbocharged. Although the use of the turbocharger had improved the steady state performance of the engine, the transient behaviour was often quite unacceptable, because of the inability of the turbocharger to supply an adequate amount of air to the engine during transients. Thermodynamists were able to represent empirically the various interactions between the turbocharger and engine, and these models were used extensively to simulate the load acceptance of turbocharged diesel engines. The emergence of these models, (which are referred to as quasi-linear models [2.14-2.16]), marked a major step forward in engine simulation - and a major increase in model detail and complexity.

The emergence of powerful digital computers in the late 1950's made it possible to solve much more fundamental engine models based on the thermodynamic and fluid-dynamic processes which occur in the cylinders and manifolds of an engine, - the so called "filling and emptying" models. The early models [2.18-2.20] represented the behaviour of an engine when it is supplied with fuel at a steady rate, and is rotating at a constant speed; consequently they did not

represent the dynamics of the fuel injection system, or the dynamics of the engine load-system. Although these models had little practical value to control engineers, the way in which they represented the behaviour of the gas in the engine made an excellent foundation from which a very detailed dynamic model of the engine could be developed. The first such model was developed by Marzouk [2.21,2.22], who extended the filling and emptying model of the gas in the engine to include representation of the engine-load system, governor and fuel injection system, thereby completing the engine speed control loop and enabling speed transients to be studied. The accuracy and usefulness of this model has been demonstrated in a number of "cause and effect" studies of factors influencing the load acceptance of turbocharged engines [2.23,2.24].

2.3 Review of Engine Models

Perhaps the most important decision to be made when selecting an engine model to use, is to decide whether the model must be able to represent the response of the engine to large changes in control inputs and disturbances (which will require a non-linear model), or whether it will suffice to represent the response to small changes about a chosen operating point, for which a linear model should be adequate. Non linear models are difficult to solve analytically and normally require use of numerical methods involving extensive computations; conversely, many standard mathematical techniques exist to evaluate the behaviour of linear equations and therefore linear models are more convenient to use.

Another important choice to be made is whether to use a physical model or black box model. A physical model consists of a set of mathematical equations which describe how the various relevant laws of physics constrain the system behaviour. Thus a physical model of an engine is derived from the direct application of the laws of thermodynamics, fluid dynamics and mechanics to each of the processes going on inside the engine. The mathematical equations describing these processes are expressed in terms of physical quantities such as engine speed, acceleration, temperature, volume etc. Such models are very versatile and can be used to evaluate system performance for a variety of configurations without the need to build any hardware. Black box modelling techniques have wide application and are generally referred to as "system identification" techniques [2.25]. Basically, the objective is to identify the mathematical relationships which are observed to exist experimentally between the output of a system and its input(s). If there are a number of outputs to be examined, then a separate black box model is produced for each output. Most system identification techniques can only result in a linear model and therefore can only represent the behaviour of an engine for small changes about the chosen operating point. However, considerable effort is being devoted to extend the technique of black box modelling so that the performance of non-linear systems can be studied when subjected to much larger changes in operating conditions [2.26].

2.3.1 Linear Engine Models

The linear models which are described below represent the speed response of an engine to changes in fueling, irrespective of whether the engine is naturally aspirated or turbocharged. Although the diesel engine is far more complex than these models imply, they are widely used in the initial design phase of engine speed controllers, and in simple simulations of power transmissions which include a diesel engine. The principle variables which are represented in the models are fuel rack position, engine brake torque and engine speed. However, there is no reason why the models should not be extended to represent other aspects of engine behaviour such as boost pressure or turbocharger speed, if knowledge of these variables is required.

2.3.1.1 Physical Linear Engine Models

The two physical models described below differ primarily in the manner in which they represent engine torque production, one representing torque production as a continuous process, and the other as a sequence of separate torque pulses - which in fact, it is.

- o **Continuous time domain model.** The simplest representation of the dynamic behaviour of an engine is that shown in Figure 2.1. The model approximates the torque production of an engine to a continuous process and assumes that the mean level of engine torque production is proportional to the fuel

rack position. The engine and load system dynamics are represented by the inertia equation, as is shown in the figure.

- Sampled data model. In reality an engine cannot begin to respond to a change in fueling until a short time interval has elapsed which depends upon the instant at which the fuel rack is moved and the time of the next injection of fuel. Also, because the fuel is injected into each combustion chamber once per engine cycle, the engine torque is generated as a series of pulses, rather than continuously. The effect of discontinuous fueling on the behaviour of a diesel engine has been extensively investigated [2.7-2.9]. These studies have shown that since the fuel is injected in a time which is short compared with the duration of the resulting torque pulse, the response may be calculated using the theory of sampling. Thus fuel injection in the engine can be represented by a sampler located after the fuel rack, as is shown in Figure 2.2. The engine torque pulses resulting from the injections of fuel can be approximated to by an appropriate hold function. For instance, a zero order hold is appropriate for a four cylinder engine, as is shown in Figure 2.3.

2.3.1.2 Black Box Linear Engine Models

Black box models of a diesel engine are derived from the application of standard mathematical procedures such as correlation, least squares and Fourier analysis to the input(s) and output(s) of

the engine. Consequently, a knowledge of the engine input and output time histories is required which must, of course, be obtained experimentally using the engine. Some system identification methods produce a black box model from a knowledge of the behaviour of the engine under normal operating conditions. However, to obtain a more accurate model, it is usual to perturb the input(s) to the engine with a special test signal (such as a pseudo random binary sequence) and to record its effect at the engine output(s). Depending upon how the time histories are analysed, parametric models of the engine can be obtained, such as z-transform models, or non-parametric models, such as the system frequency response. System identification techniques are normally applied to a diesel engine to obtain a model which represents the speed response of the engine to changes in fueling.

2.3.2 Non-Linear Engine Models

In many applications, the linear models described above are quite inadequate because they are unable to represent the response of the engine to large variations in control inputs or external disturbances and, for these cases, a non-linear engine model must be used. Non-linear engine models range from relatively simple models, which include a large empirical content, to the complex fundamental models which take into account the physical processes occurring in the individual cylinders and manifolds of the engine. Obviously the accuracy, usefulness and amount of computation required for these models varies greatly. The simpler models require little computation and will normally only represent a particular engine

design (because of the empirical data built into the model which relates to that engine). Conversely, the fundamental models, which require extensive computation, have much wider application, and can even be used to represent the behaviour of conceptual engines which are still in their design stage.

2.3.2.1 Quasi-Linear Models

Most current dynamic engine simulations are based on a quasi-linear type of engine model [2.27] and have the advantages of simplicity and a correspondingly short computer run time. The most widely reported models of this type are those which were developed by the University of Manchester Institute of Science and Technology [2.14,2.15].

Essentially, quasi-linear engine models link the measured steady state thermodynamic and fluid-dynamic performance of an engine with physical models of the engine and turbocharger speed dynamics and the governor in the manner shown in Figure 2.4. The steady state engine performance data is used to represent the behaviour of variables such as exhaust gas temperature and scavenge flow as the operating condition of the engine changes.

The principal weakness of quasi-linear models lies in their heavy reliance on this empirical data, obtained experimentally with the engine operating under steady conditions and therefore not truly representative of engine conditions during a transient. This, together with other simplifications concerning the behaviour of the engine, limit the accuracy of the models [2.17]; even so they can

provide a useful insight into the operation and interactions in a turbocharged diesel engine, and have been used to study their transient performance, with some success [2.28].

2.3.2.2 Filling and Emptying Models

Filling and emptying models derive their name from the way in which they follow the successive "filling and emptying" phases of the engine cylinders and manifolds with gas, as the engine operates. These models are very detailed, being based on the fundamental thermodynamic and fluid-dynamic processes which occur in the individual cylinders and manifolds of the engine during successive phases of engine operation. Because of this, unlike the simpler models, the operation of a filling and emptying model bears a very close resemblance to the operation of a real engine, as follows:-

The inlet manifold draws air from the atmosphere if it is a naturally aspirated engine, or air is supplied from the compressor model if the engine is turbocharged. Heat transfer takes place between the gas and the surroundings, through the manifold walls.

Each engine cylinder is supplied with air from the inlet manifold, and ejects exhaust gas to the exhaust manifold when the appropriate valves are open. Towards the end of its compression stroke, a cylinder receives a shot of fuel from the injector, with a resulting heat release during the combustion phase. Work is transferred to and from the piston

and heat transfer takes place between the gas and the combustion chamber walls.

The exhaust manifold ejects hot exhaust gas to the atmosphere if the engine is naturally aspirated, or to a turbine if the engine is turbocharged; again, heat is exchanged with the surroundings.

It is the ability of filling and emptying models to accurately represent the changing conditions of the gas in an engine which makes them particularly useful for fundamental engine design studies, such as turbocharger matching, sensitivity analysis and component stress calculations. The models are also useful to the control engineer since, unlike the simpler models, they are able to represent the way in which all important aspects of engine behaviour (apart from emissions) respond to changes in any control input, or external disturbance.

The basis of the model is to represent the engine as a number of inter-connected thermodynamic control volumes which represent the engine manifolds and cylinders. Thus the single cylinder engine shown in Figure 2.5 is represented by three control volume models, one each for the inlet manifold, cylinder and exhaust manifold. State equations are derived for each control volume model and represent the rate of change of temperature, the rate of change of fuel-air ratio and the rate of change of mass of gas in the control volume. Evaluation of the state equations requires the use of sub-models which represent various physical processes which occur in the engine, including combustion, heat transfer, valve flow and the

turbocharger. Solution of the model proceeds in small steps of crankshaft position (or time) and yields the changing conditions of the gas in the individual cylinders and manifolds of the engine.

In a model which represents the dynamic response of an engine, the filling and emptying model of the gas in the engine cylinders and manifolds is just one element in the total engine model. The inputs to the gas model are a supply of fuel and air, from which engine brake torque and then engine acceleration are calculated as is shown in Figure 2.6.

Before leaving the subject of filling and emptying engine models, brief mention will be made of the modelling of emissions. With legislation already announced governing the maximum permitted level of exhaust gas pollution, the control of emissions is high on the priority list of engine manufacturers and there is now a great need for models capable of predicting them. Unfortunately, the development of models capable of doing this is exceedingly difficult, and represents a major technical challenge - as does the control of emissions themselves.

Despite the relative complexity of filling and emptying models they still contain major approximations concerning the state of the gas in the engine cylinders and manifolds. In particular, thermodynamic equilibrium is assumed, which means that no local variations in gas composition, temperature, pressure etc are assumed to exist within the volume. However, in a fundamental study of emission formation, it is essential to model local variations in gas quantities in the fuel spray and combustion areas; thus filling and

emptying models in their present form are inadequate for this application. Very fundamental multi-zone combustion models are being developed to overcome these limitations [2.29], but are, as yet, in an early state of development, and not accurate enough for engine performance simulation [2.30].

Fortunately the needs of the control engineer are not so exacting as to require the use of the complex models required by thermodynamics. Simpler experimentally based models which reflect the overall change in emission formation in response to changes in the control inputs and external disturbances are usually adequate. These simpler emission models are compatible with existing filling and emptying models and Watson [2.27] has already incorporated one such model into an engine simulation. This model predicts the smoke opacity of the exhaust gases [2.31]; other models to predict particulates, nitrous oxide and hydrocarbon emissions have been proposed [2.32,2.33]. Of course, the addition of the emission models, will further increase the already considerable computational requirements of filling and emptying models.

2.4 Summary

A brief review of the historical development of engine models and a description of different approaches to dynamic modelling has been given in this chapter. This shows that the trend in engine modelling has been from simple linear models to much more complex models which are capable of representing most aspects of engine behaviour. Thus the engineer is now in the fortunate position that,

irrespective of what the control objective may be, it is more than likely that an engine model will have been developed suitable for the task. Nevertheless, the trend towards the use of more complex engine models seems set to continue, since, whilst the fundamental filling and emptying models in use today adequately represent most aspects of engine performance, they are severely limited in their ability to predict emissions.

2.5 References

2.1 C Maxwell.
On Governors.
1868, Proc. Roy. Soc. 16, p270.

2.2 E J Routh.
On the Stability of Motion.
1877, Macmillan, London.

2.3 D B Welborne, D K Roberts, R A Fuller.
Governing of Compression Oil Engines.
1959, Proc. I. Mech. E 173 No 22.

2.4 J Z Bujak.
The Variable Speed Hydraulic Governor.
1945, Proc. I. Mech. E 153.

2.5 A G Massey, G Oldenburger.
Scientific Design of a Diesel Governor.
1958, ASME 58-OGP-11.

2.6 G W Taylor.
Development of a Speed and Load Sensing Governor.
1960, ASME 6U-OGP4.

2.7 D E Bowns.
The Dynamic Transfer Characteristics of Reciprocating Engines.
1971, Proc. I. Mech. E 185.

2.8 P A Hazell, J O Flower.
Sampled Data Theory applied to the Modelling and Control Analysis of Compression Ignition Engines - Part I.
1971, Int. J. Control p 549-562.

2.9 P A Hazell, J O Flower.
Sampled Data Theory applied to the Modelling and Control Analysis of Compression Ignition Engines - Part II.
1971, Int. J. Control p 609-623.

2.10 P E Wellstead, C Thiruarooran, D E Winterborne.
Identification of a Turbocharged Diesel Engine.
1978, Proc. IFAC 7th World Congress, Helsinki, Finland.

2.11 C Thiruarooran.
Identification and Control of a Turbocharged Diesel Engine.
1978, PhD Thesis University of Manchester.

2.12 J O Flower, G P Windett.
Dynamic Measurements of a Large Diesel Engine using prbs Techniques.
Part I. Development of Theory for Closed Loop Sampled Systems.
1976, Int. J. Control, vol 24 p 379-392.

2.13 J O Flower, G P Windett.
Dynamic Measurement of a Large Diesel Engine using prbs Techniques.
Part II. Instrumentation, experimental techniques and results.
1976, Int. J. Control, vol 24, p 393-404.

2.14 J D Ledger, S Walmsley.
Computer Simulation of a Turbocharged Diesel Engine Operating under Transient Load Conditions.
1971, SAE 710177.

2.15 J D Ledger, R S Benson, N D Whitehouse.
Dynamic Modelling of a Turbocharged Diesel Engine.
1973, I. Mech. E Conf. on Engine Performance Modelling, London.

2.16 D E Bowns, P R Cave, M R O Hargreaves, F J Wallace.
Transient Characteristic of Turbocharged Diesel Engines.
1973, Proc. I. Mech. E CP15.

2.17 D E Winterborne, C Thiruarooran, P E Wellstead.
A Wholly Dynamic Model of a Turbocharged Diesel Engine for Transfer Function Evaluation.
1977, SAE 770124.

2.18 G L Borman.
Mathematical Simulation of Internal Combustion Engine Process.
1964. PhD Thesis, University of Wisconsin.

2.19 K J McAulay, W Tang, S K Chen, G L Borman, P S Myers, O A Uyehara.
Development and Evaluation of the Simulation of the Compression-Ignition Engine.
1965, SAE 650451.

2.20 E E Streit, G L Borman.
Mathematical Simulation of a Large Turbocharged Two-Stroke Diesel Engine.
1971, SAE 710176.

2.21 M Marzouk.
Simulation of Turbocharged Diesel Engines under Transient Conditions.
1976, PhD Thesis, University of London (Imperial College).

2.22 N Watson, M Marzouk.
A Non-Linear Digital Simulation of Turbocharged Diesel Engines Under Transient Conditions.
1977, SAE 770123.

2.23 M Marzouk, N Watson.
Load Acceptance of Turbocharged Diesel Engines.
1977, I. Mech. E C54/78.

2.24 N Watson.
Transient Performance Simulation and Analysis of Turbocharged Diesel Engines.
1981, SAE 810338.

2.25 K J Astrom, P Eykoff.
System Identification - A survey.
1970, Automatica.

2.26 S A Billings.
Identification of Non-Linear Systems - A survey.
Nov 1980, IEE Proc. Vol. 127 pt D No 6.

2.27 N Watson.
Dynamic Turbocharged Diesel Engine Simulator for Electronic Control System Development.
1984, J. of Dynamic Systems, Measurement and Control, Vol 106.

2.28 D E Winterborne, R S Benson, C D Closs, A G Mortimer.
A Comparison between Experimental and Analytical Transient Test Results for a Turbocharged Diesel Engine.
1976, Proc. I. Mech. E Vol 190.

2.29 M Megerdichian, N Watson.
Mixture Preparation and Heat Release in Diesel Engines.
1978, SAE 780225.

2.30 N Watson, M S Janota.
Development and Application of a Comprehensive Turbocharged Diesel Engine Model.
1980, Un.I.C.E.G King's College, London.

2.31 W Bryzik, C O Smith.
Relationship between Exhaust Smoke Emissions and operating variables in Diesel Engines.
1977, SAE 770718.

2.32 G Greeves, C H T Wang.
Origins of Diesel Particulate Mass Emissions.
1981, SAE 810260.

2.33 D R Nightingdale.
A Fundamental Investigation into the problem of NO Formation in Diesel Engines.
1975, SAE 750848.

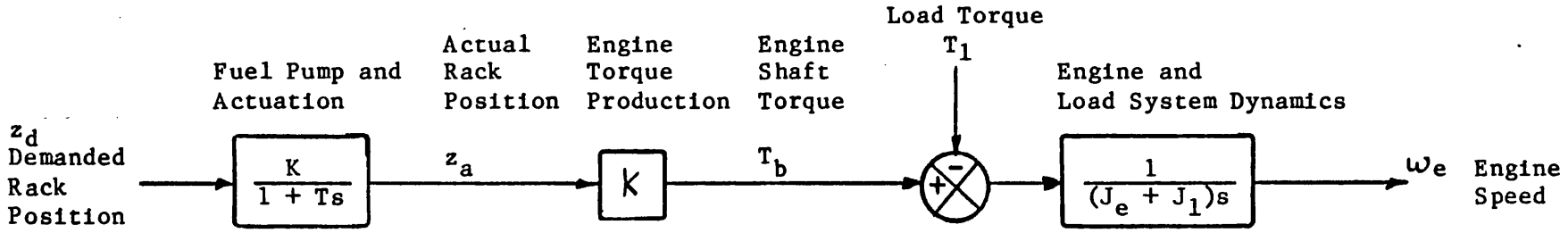


Figure 2.1 Linear Continuous Time Domain Engine Model

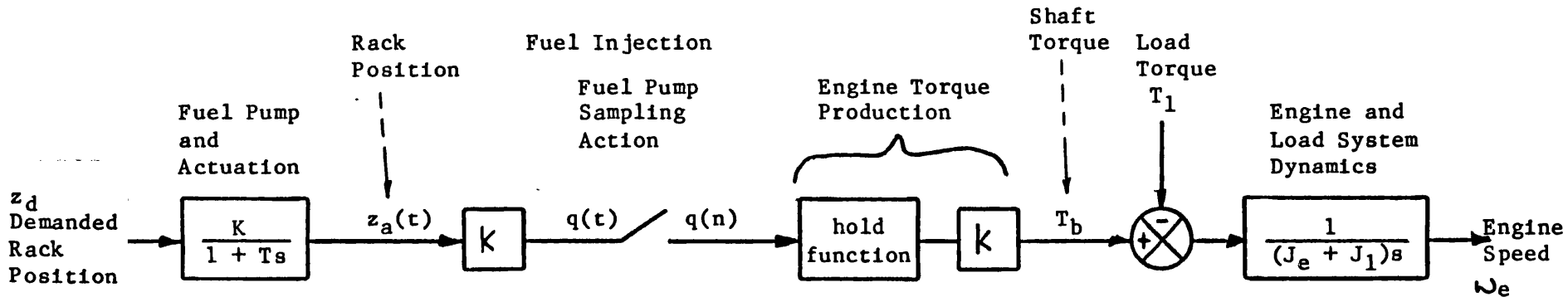


Figure 2.2 Linear Sampled data Engine Model

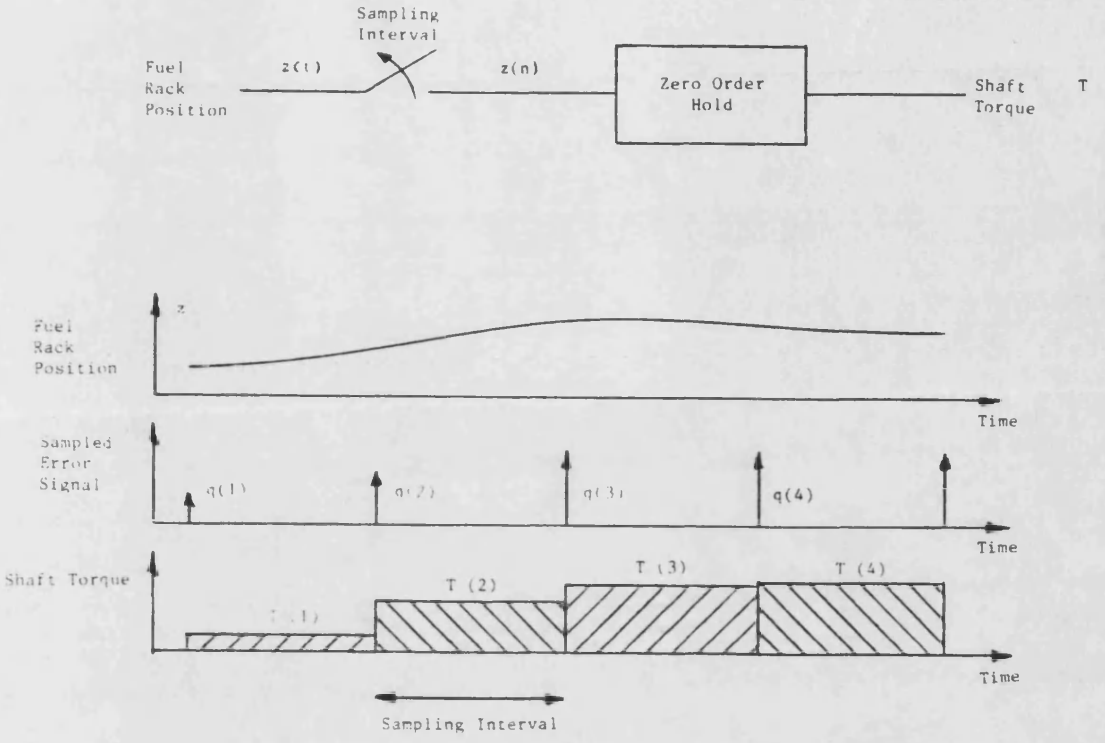


Figure 2.3 Sampled Data Model of a 4 Cylinder Engine

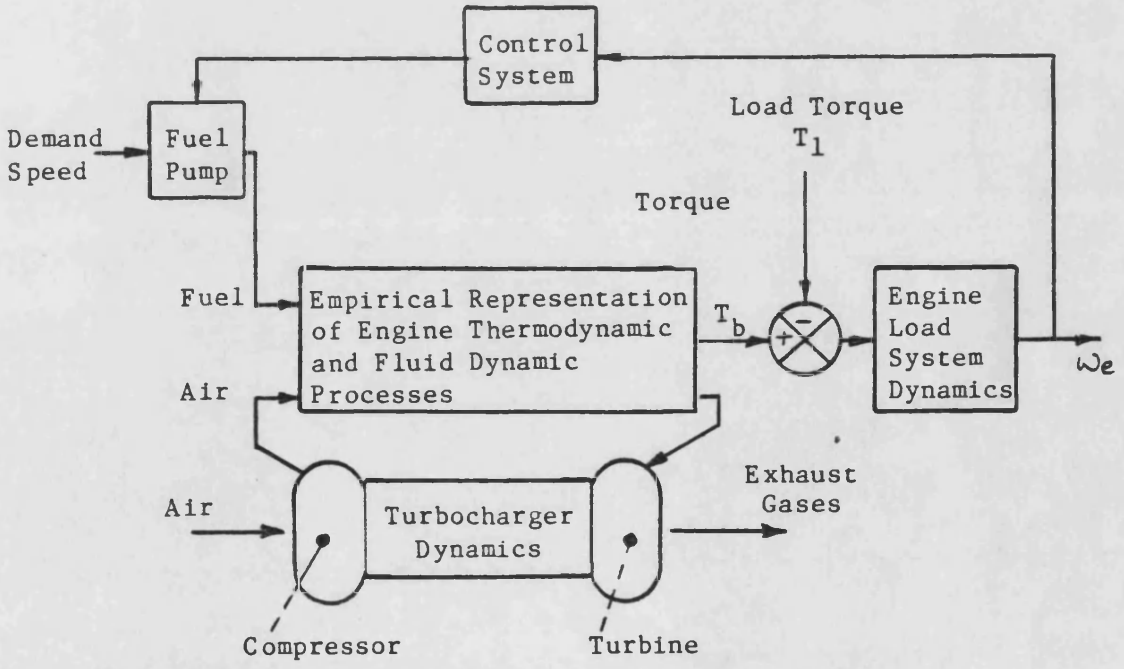


Figure 2.4 Block Diagram Representation of Quasi-Linear Engine Model

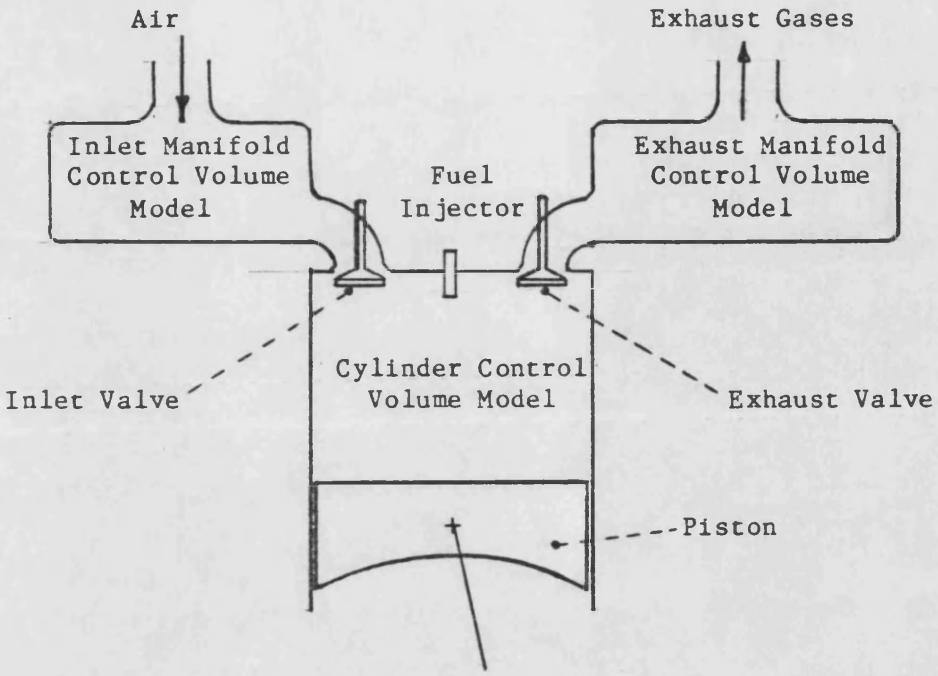


Figure 2.5 Schematic of Single Cylinder Engine

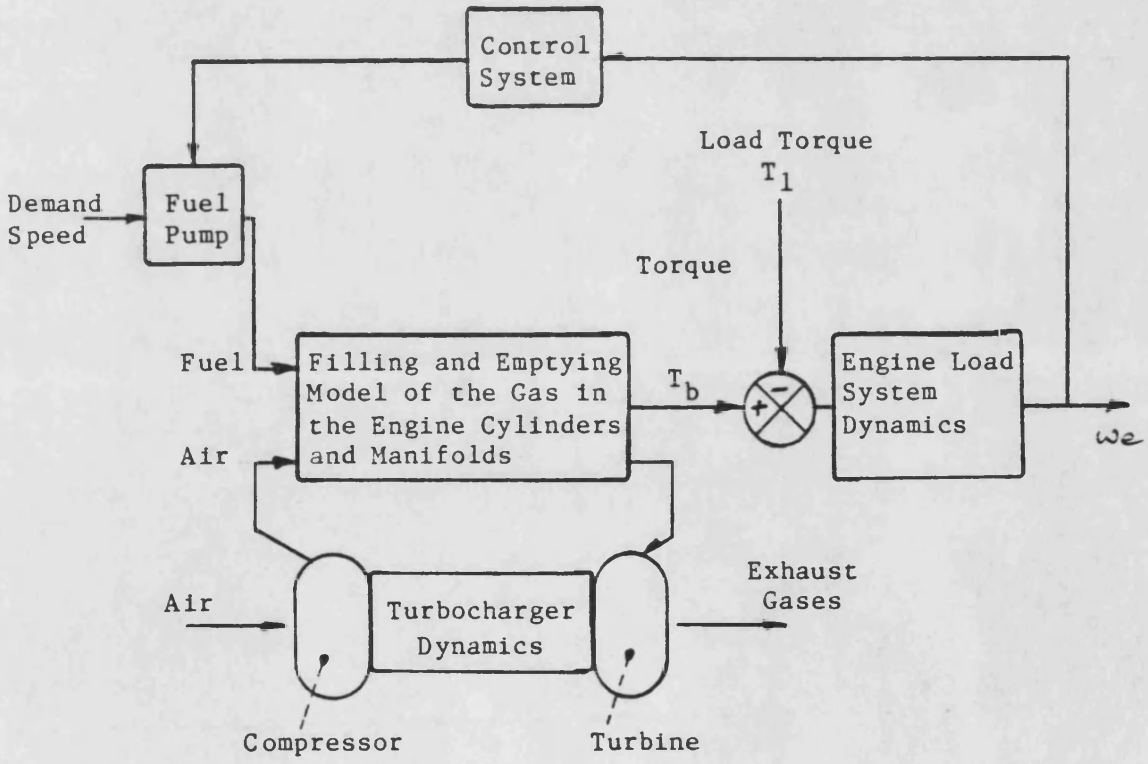


Figure 2.6 Filling and Emptying Model of a Turbocharged Diesel Engine

CHAPTER 3

3.1 The Engine Modelled

It was considered very important to assess the performance of the parallel solution technique of the engine model developed during this programme of research, by applying it to the simulation of a real engine, and the engine modelled is described in this chapter. The School of Mechanical Engineering at the University of Bath has a number of diesel engine test beds, ranging from simple, single cylinder engines, to much more complex multi-cylinder turbocharged truck engines. Modelling a multi-cylinder engine provides a more realistic opportunity to assess the practicability and benefits of applying parallel processing to engine modelling. It was decided therefore, to model the experimental Leyland TL11 engine, since with the exception of its turbocharger, it is typical of the vast majority of engines used in truck transportation. In addition, the engine was available for experimental work, should that be necessary.

3.2 Description of the Engine

The engine modelled is based on a Leyland TL11 diesel engine, which was designed in the late 1970's primarily as a power unit for trucks and buses. It has six "in line" cylinders having a total displacement of 11.1 litres. The engine operates on a four stroke power cycle and generates a maximum power of 190kW when at its

maximum speed of 2100 rpm. Figures 3.1a and 3.1b show the engine installed in the test cell. A detailed description of the engine is given in Reference 3.1 and Appendix A1 lists those physical features of the engine which are required to model its behaviour, using a filling and emptying model.

Fuel is injected directly into the engine combustion chambers from the fuel pump (shown in Figure 3.2), which is driven from the engine crankshaft. The fuel pump has two controls: one control determines the quantity of fuel injected into the combustion chamber (rack control), and the other determines the instant at which fuel injection starts (timing control). Separate hydraulic actuators controlled in closed loop, are used to position the rack and the timing controls, as shown in Figure 3.3.

The engine is equipped with a variable geometry turbocharger (shown in Figure 3.4), which gives some control over the supply of air to the engine. The turbocharger supplies the engine with air from a conventional compressor section, with the variable geometry capability being provided by the turbine section. The compressor is in the housing shown on the left of Figure 3.4, and the turbine in the housing on the right. The precise manner in which the variable geometry behaviour has been obtained is confidential to the company who supported the development of the turbocharger, but it is understood that it depends upon the use of a variable area nozzle ring arrangement, located immediately upstream of the turbocharger turbine. A hydraulic actuator (Figure 3.3), controlled in closed loop is used to position the nozzle ring, and can adjust the turbine nozzle area from 0% (no reduction in turbine nozzle area) to a

maximum of 50% (50% reduction in nozzle area). Part of the actuating mechanism can be seen in the photograph, Figure 3.4, (between the compressor and turbine housings). The turbine is supplied with exhaust gas through a single entry casing fed from two separate exhaust manifolds, each exhaust manifold being supplied with gas from three engine cylinders. The use of a single entry turbine casing is quite unusual, and was adopted because of flow restriction caused by the variable geometry mechanism [3.1].

The extent to which the variable geometry turbocharger is able to control boost pressure can be seen from the steady state performance map of the turbocharger compressor, given in Figure 3.5. The map shows the steady state operating conditions for the compressor over the speed range of the engine, when the engine is producing its limiting torque. These operating conditions are shown for the two extremes of geometry (0 and 50% nozzle area reduction), and the area between them defines the range over which boost pressure can be controlled. For example, under limiting torque conditions, and an engine speed of 1500 rpm, the boost pressure can be controlled from about 1.7 bar (at a turbocharger speed of 65,000 rpm), to about 2.2 bar (at a turbocharger speed of 85,000 rpm).

A hydraulic dynamometer, shown in Figure 3.6, is used to absorb the power developed by the engine. It can be set up to absorb power at constant torque, constant speed, or at a torque which is proportional to the square of the engine speed ("windage").

The engine and turbocharger are extensively instrumented. All the signals are routed from the test cell into the control room, from where the engine is driven, and are available on the instrumentation panel (shown in Figure 3.7), for display and/or recording.

3.3 Discussion of Engine Behaviour

Figure 3.8 is a comprehensive diagram of the way in which the engine system produces power. Each block represents a particular physical process, or sub-system, with the control inputs (ie fuel rack position, fuel injection timing, and turbine nozzle area) being shown to the left of the diagram. It can be seen from Figure 3.8 that two internal feedback loops exist within the system, one in the fuel supply and one in the air supply.

- Fuel supply feedback loop: the engine fuel pump is driven directly from the engine crankshaft by a mechanical drive and if the fuel rack position is not changed, the quantity of fuel supplied to the engine changes, more or less, in proportion to engine speed. Consequently, the fuel rack position has to be controlled by a speed governor.

- Air supply feedback loop: during the engine induction stroke, the turbocharger supplies air to the combustion chamber. In the subsequent exhaust stroke, the hot exhaust gas is expelled from the combustion chamber into the exhaust manifold, where it is used to drive the turbocharger turbine - thus completing the feedback loop. The existence of this loop is the reason

for the poor transient response of the engine, - the combined effect of the turbocharger inertia and compressibility of the gas causing the supply of air to the engine during a transient to lag significantly behind that which is required.

It has already been mentioned that, because the diesel engine receives injections of fuel at discrete instants of time, the engine torque is developed as a series of pulses. An important consequence of this is that the engine is only influenced by the position of the fuel rack at those instants at which fuel is injected. The interactions which occur between fuel rack position and engine torque, fuel rack position and boost pressure, turbine nozzle area and boost pressure and between turbine nozzle area and engine torque, are shown on Figure 3.8. This shows that the engine behaves as a quite complex multivariable system, since engine torque and boost pressure both change when either the fuel rack position, or the turbine nozzle area, is changed. Many of these relationships between the system input(s) and output(s) are non-linear, and consequently both the engine and turbocharger have dynamic characteristics which depend upon the engine operating load and speed.

The complex interactions which occur between control inputs and engine outputs make the study of the engine of considerable interest from a control point of view and pose a challenging multivariable control problem. However, although the engine model used must be able to represent these complex interactions, the design of such a controller would be an irrelevance to the proper pursuit of this research - and no attempt was made to do so.

3.4 References

3.1 E W Roberts.
Variable Geometry Turbocharging Optimisation and Control.
1984, PhD Thesis, University of Bath.

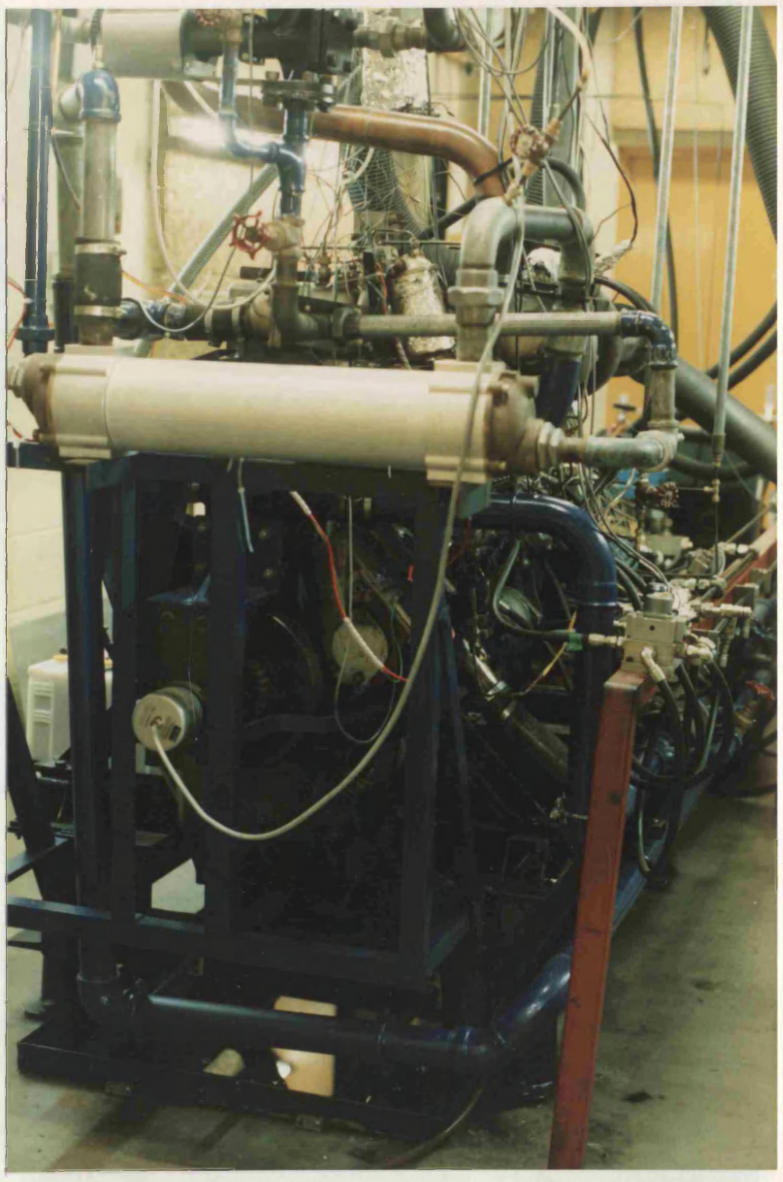


Figure 3.1a TL11 Engine Test Bed Installation

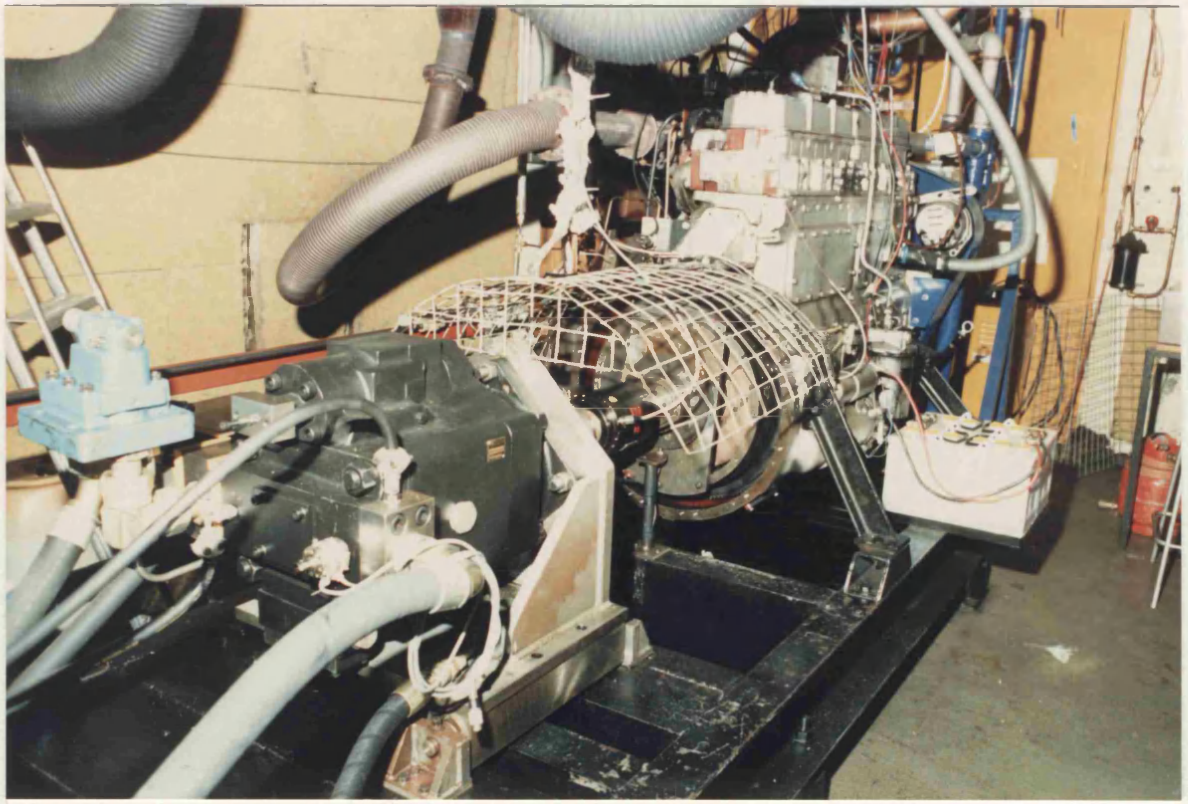


Figure 3.1b TL11 Engine Test Bed Installation

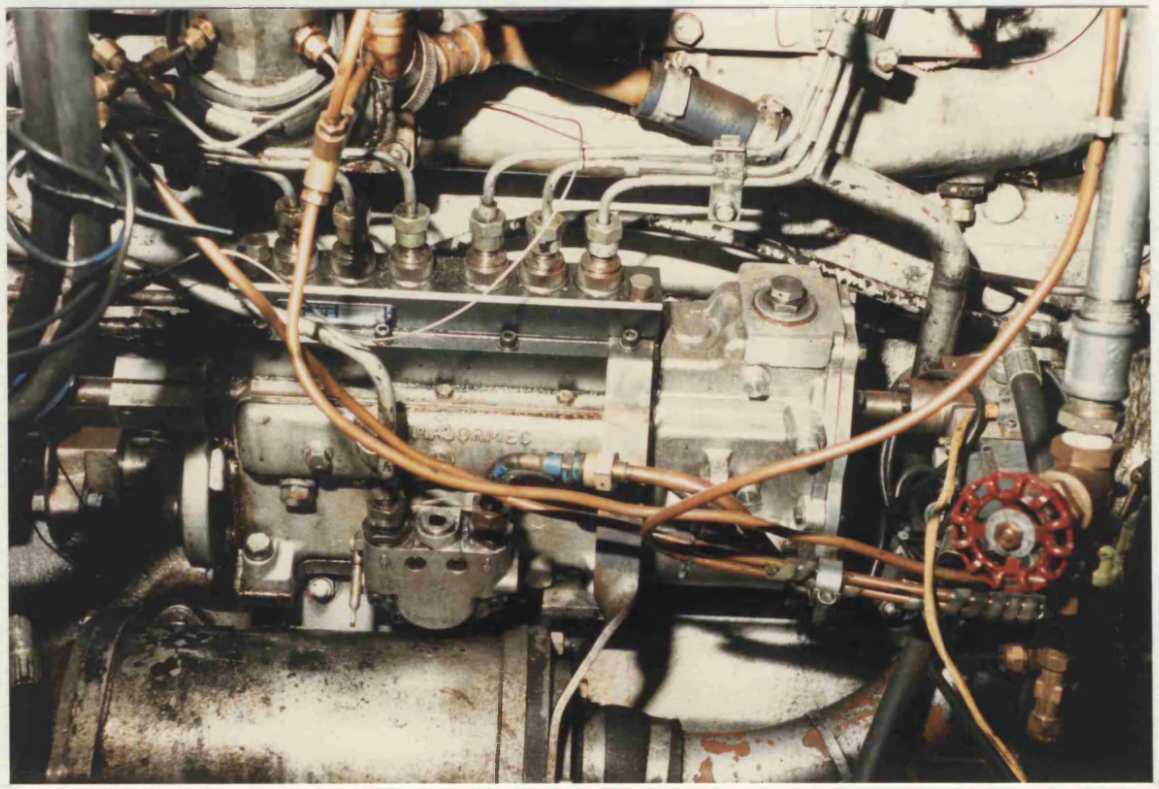


Figure 3.2 Fuel Pump

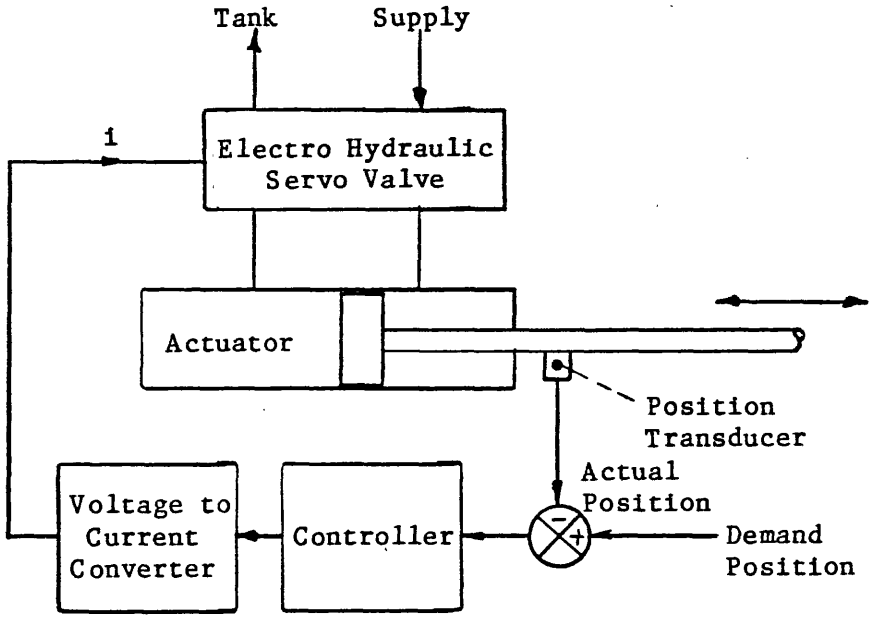


Figure 3.3 Engine Control Input Actuation System

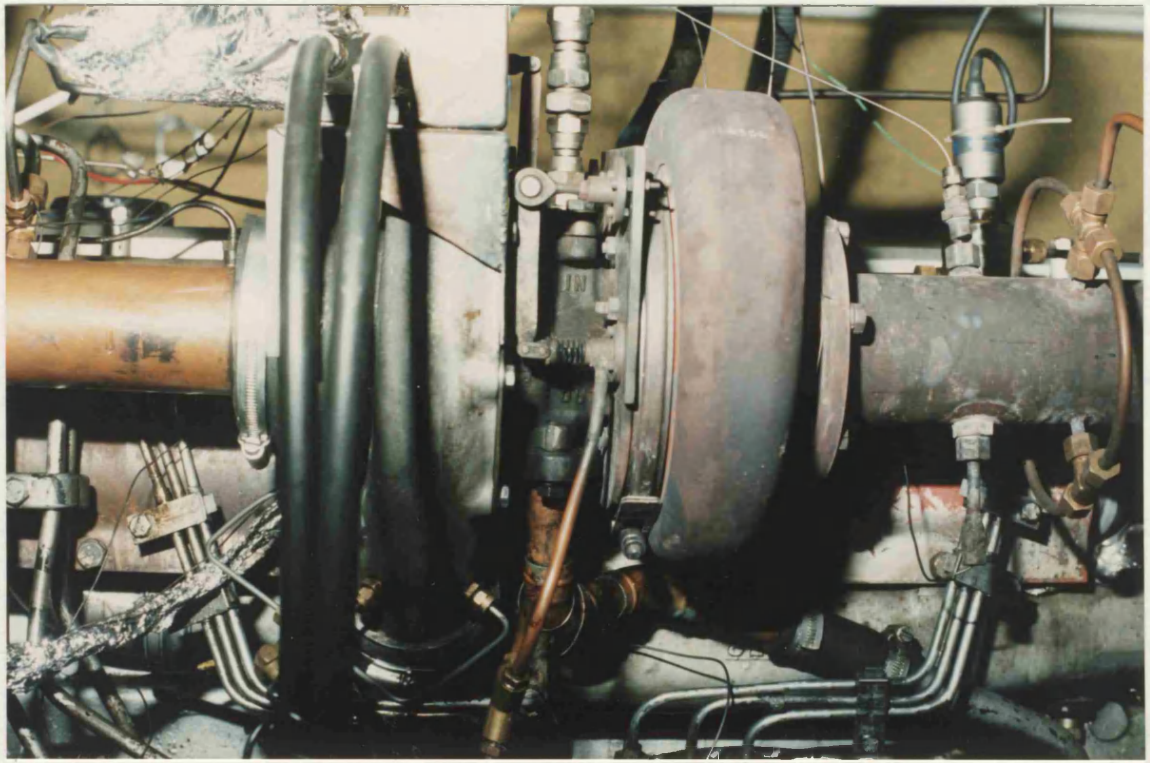


Figure 3.4 Variable Geometry Turbocharger

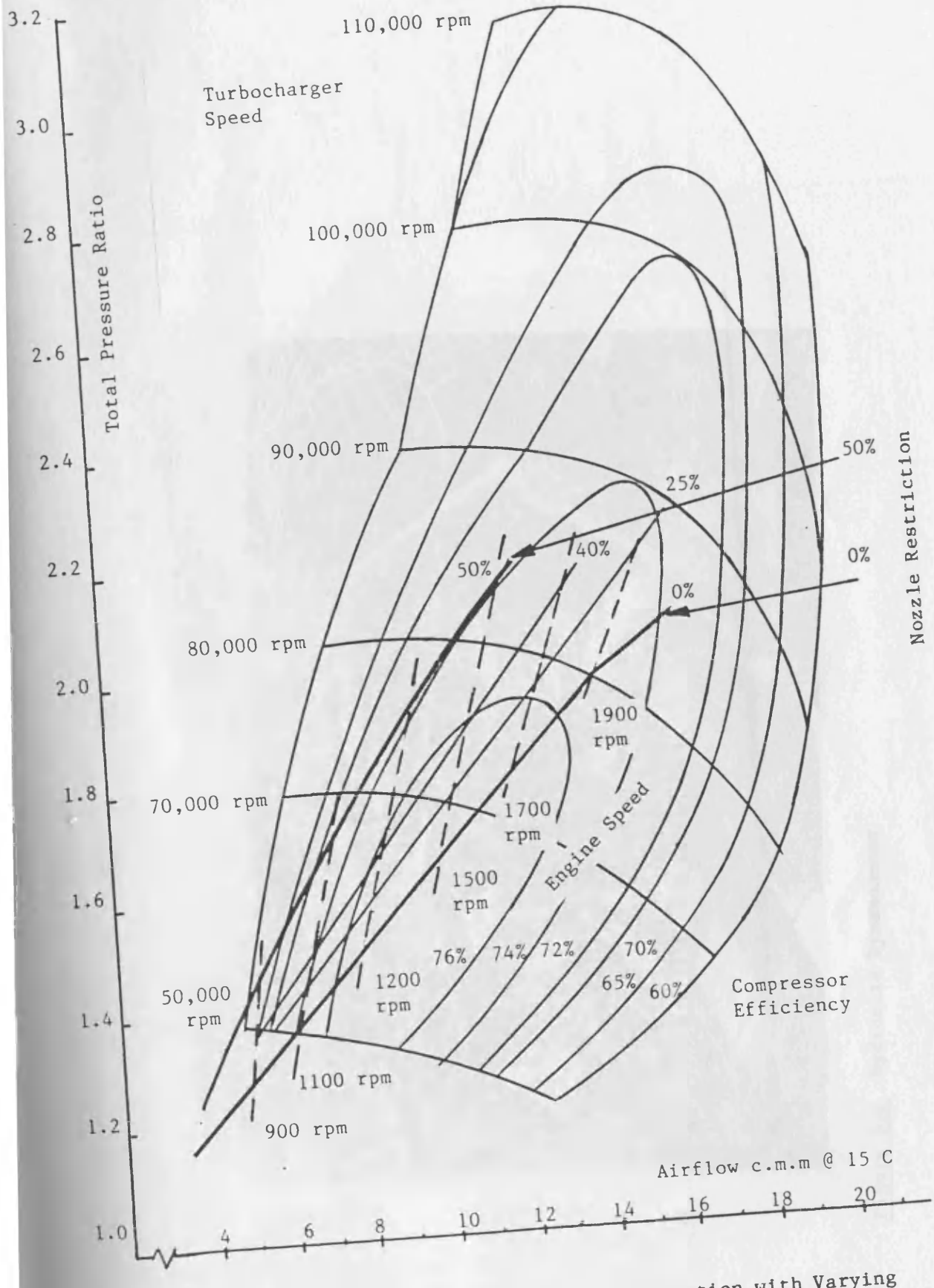


Figure 3.5 Compressor Limiting Torque Curve Operation with Varying Turbine Area Reduction

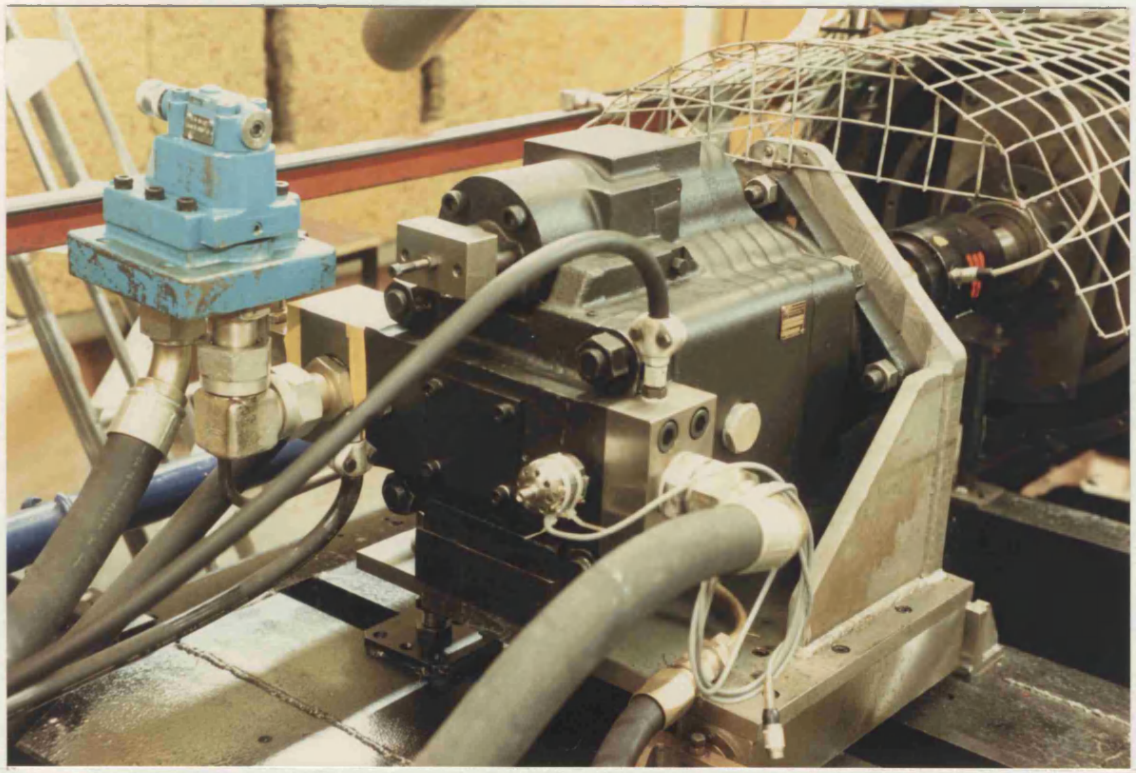


Figure 3.6 Hydraulic Dynamometer

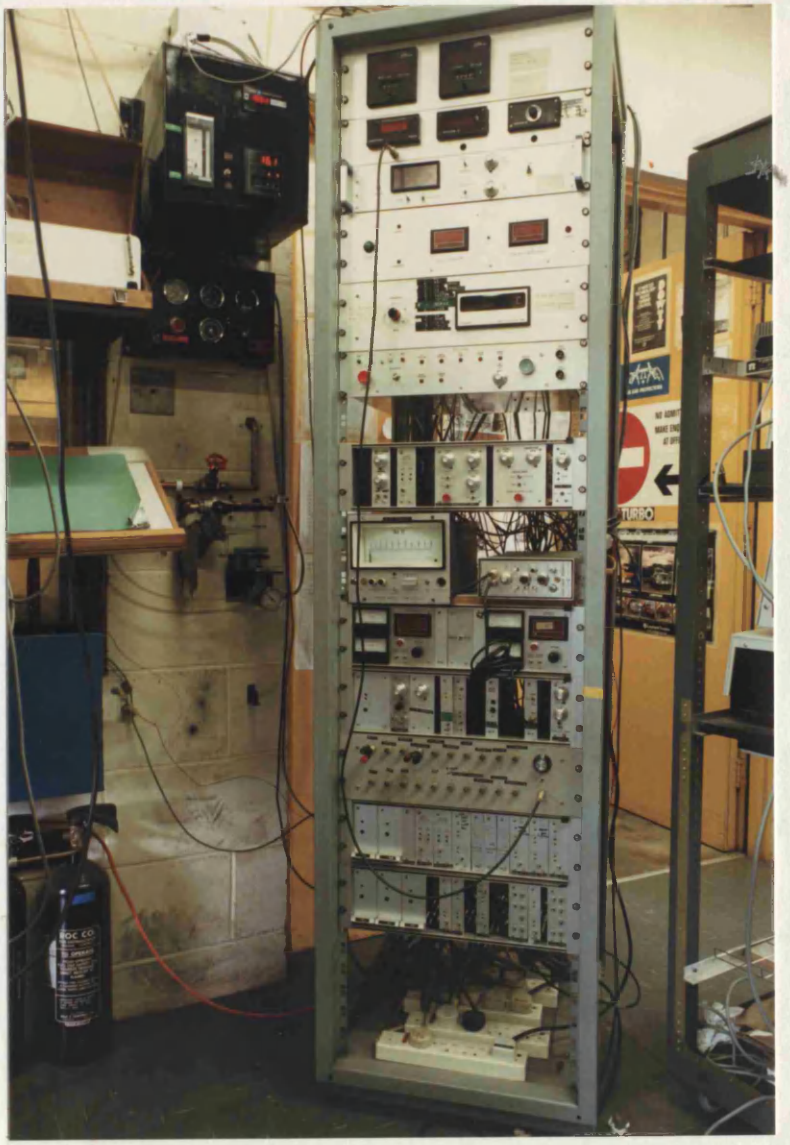


Figure 3.7 Instrumentation Panel

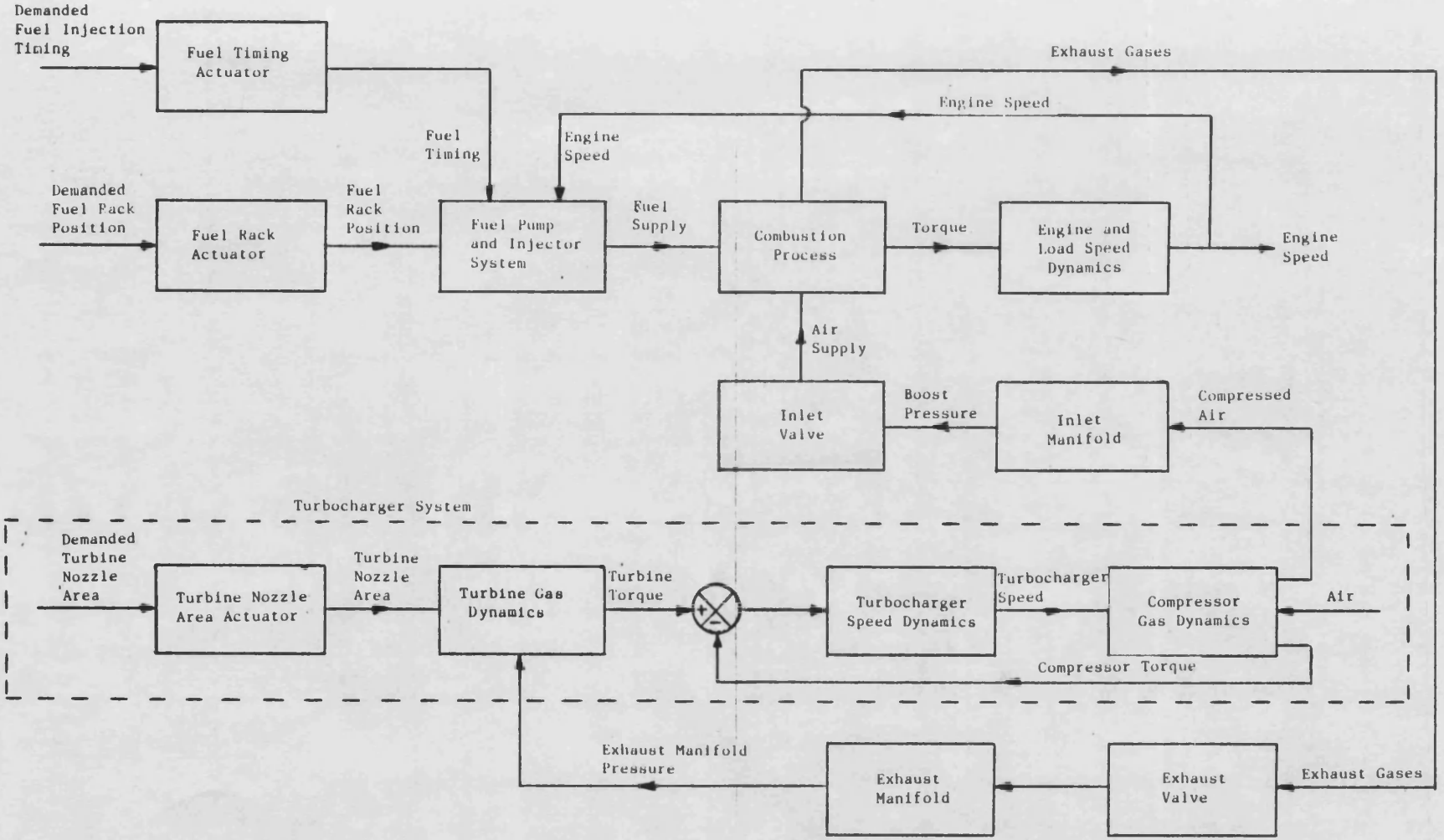


Figure 3.8 Block Diagram Representation of the Behaviour of the Experimental TL11 Engine.

CHAPTER 4

4.1 Description of the Engine Model

The complex multivariable interactions which occur between the fuel rack control input, turbine nozzle area control input and the engine outputs, make the experimental TL11 engine a particularly interesting and challenging engine to model. More importantly though, it provides a good opportunity to assess the benefits of applying a parallel solution to a filling and emptying model. The model is a development of the efforts of many individuals, with perhaps most notable contributions being made by Borman et al. [4.1-4.3] who developed the basis of the filling and emptying model type, and Marzouk [4.4] who extended the model so that it could represent the transient response of the engine. The overall engine model, consists of the fundamental filling and emptying model of the gas behaviour in the engine cylinders and manifolds (described in Section 4.2 to 4.5), together with models to represent the dynamic behaviour of the engine control inputs and the speed dynamics of the engine load system (described in Section 4.6). It should be noted that the model described is specifically the one used to represent the experimental TL11 engine.

4.2 Modelling Assumptions

The diesel engine is a highly complex device, and it is inevitable that various simplifying assumptions have to be made

before its behaviour can be described mathematically. Most of the assumptions which have been made are mentioned at appropriate points in the discussion. However, those fundamental assumptions which concern the modelling of the gas in the engine and which are vital to the development of the model, are presented below.

- o perfect mixing: it is assumed that whenever mass flow occurs between two control volumes, the resulting mixture is, at all times, homogeneous.

- o thermodynamic equilibrium: it is assumed that the content of a control volume is always at uniform temperature and pressure.

- o semi-perfect gas behaviour: the gas is assumed to satisfy the perfect gas equation:-

$$P.V = m.R.T \qquad (4.1)$$

  and the thermodynamic properties of the gas are assumed to be a function of gas temperature only, (ie the effect of dissociation are neglected).

- o quasi-steady flow: although gas flow entering, or leaving a control volume is non-steady, it is assumed that the processes which occur in the engine, behave under non-steady flow conditions, as they would if the flow rate at every instant were steady.

4.3 Overview of the Filling and Emptying Model

Figure 4.1 shows how the thermodynamic behaviour of the

experimental TL11 engine is represented using a filling and emptying model. Central to the model, are the six thermodynamic control volumes, each representing an engine cylinder. The inlet manifold, and the two exhaust manifolds are also represented by separate thermodynamic control volumes. Thus in total, nine thermodynamic control volumes are used to represent the engine.

The exhaust manifolds can be represented by simple thermodynamic control volumes, since the quantity of energy which is transferred to the turbocharger turbine by the pressure wave action via the two exhaust manifolds, is not significant in this engine. This is fortunate, since, if the influence of the pressure wave were to have been significant, then the exhaust manifolds would have had to be represented by an arrangement of pipes, and the equations for compressible unsteady flow solved. This would have resulted in a severe computational penalty, which, according to Reference 4.5, might be as high as ten.

The inlet and exhaust manifold control volumes are linked to the cylinder control volumes by the flow of gas through the appropriate valves. The inlet manifold control volume is supplied with air from the turbocharger compressor, and the two exhaust manifold control volumes eject exhaust gases to the variable geometry turbine.

4.4 Description of the Control Volume Gas State Equations

The state of the gas in a control volume can be represented at all instants by its temperature, mass and fuel/air ratio. Fuel/air

ratio is defined as the mass of burnt fuel to the mass of air in the control volume. The state equations which express the rates of change of these variables, are derived from the laws of conservation of energy, fuel species and mass, and are given below:-

- rate of change of temperature (derived from the energy equation)

$$\frac{dT}{d\theta} = \frac{1}{m.\frac{\partial u}{\partial T}}\left[\sum \frac{dm_i}{d\theta}.ho_i + \sum \frac{dm_o}{d\theta}.ho_o + \frac{dm_f}{d\theta}.h_{for} - u.\frac{dm}{d\theta} + \sum \frac{dQ_w}{d\theta} - P.\frac{dV}{d\theta} - m.\frac{\partial u}{\partial f}.\frac{df}{d\theta}\right] \quad (4.2)$$

- rate of change of fuel/air ratio (derived from conservation of fuel species)

$$\frac{df}{d\theta} = \frac{1+f}{m}\left[(1+f).\frac{dm_{fb}}{d\theta} - f.\frac{dm}{d\theta}\right] \quad (4.3)$$

- rate of change of mass (derived from conservation of mass)

$$\frac{dm}{d\theta} = \sum \frac{dm_i}{d\theta} + \sum \frac{dm_o}{d\theta} + \frac{dm_f}{d\theta} \quad (4.4)$$

where

f	is fuel/air ratio	suffix f	- fuel
ho	specific stagnation enthalpy	fb	- fuel burnt
m	mass	for	- formation
P	pressure	i	- in
Q_w	heat transfer	o	- out
T	temperature		
u	specific internal energy		
V	volume		
θ	crankshaft angle		

Crankshaft position is used as the independent variable, rather than time, since much of the engine operation, including that of the valves, and fuel injection is determined by crankshaft position. It is related to time by the equation:-

$$\theta = \int \omega_e . dt + \theta_o \qquad (4.5)$$

where θ_o is the angular position of the crankshaft at t=0.

The state equations expressed above are quite general and can be used to represent the state of the gas in any control volume, irrespective of whether it physically represents a cylinder, or a manifold; consequently, certain terms in the equations are not always relevant. For instance, combustion does not take place in a manifold, and only occurs in a cylinder immediately following an injection of fuel. Exclusion of irrelevant terms from the model avoids nugatory computation, but then results in separate models to represent the state of the gas in the manifolds, and in a cylinder, for each of the different phases of engine operation i.e., scavenge, induction, compression, combustion, power and exhaust. Although using individual models results in a significantly larger computer program, this is more than compensated for, by the reduction in computational time.

The models used to represent the behaviour of the gas in a cylinder during the different phases of engine operation, and in a manifold are:-

- o cylinder control volume during scavenge: inlet valve open, exhaust valve open, no combustion.

$$\frac{dm_f}{d\theta} = 0$$

substituting into the gas equations gives:-

$$\frac{dT}{d\theta} = \frac{1}{m.\frac{\partial u}{\partial T}}\left[\frac{dm}{d\theta}_{iv}.ho_{iv} + \frac{dm}{d\theta}_{ev}.ho_{ev} - u.\frac{dm}{d\theta} + \frac{dQ_w}{d\theta} - P.\frac{dV}{d\theta} - m.\frac{\partial u}{\partial f}.\frac{df}{d\theta}\right] \quad (4.6)$$

$$\frac{df}{d\theta} = \frac{1+f}{m}\left[(1+f)\left[\frac{f_{iv}}{1+f_{iv}}.\frac{dm}{d\theta}_{iv} + \frac{f_{ev}}{1+f_{ev}}.\frac{dm}{d\theta}_{ev}\right] - f.\frac{dm}{d\theta}\right] \quad (4.7)$$

$$\frac{dm}{d\theta} = \frac{dm}{d\theta}_{iv} + \frac{dm}{d\theta}_{ev} \quad (4.8)$$

- o cylinder control volume during induction: inlet valve open, exhaust valve closed, no combustion.

$$\frac{dm}{d\theta}_{ev} = \frac{dm_f}{d\theta} = 0$$

substituting into the gas equations gives:-

$$\frac{dT}{d\theta} = \frac{1}{m.\frac{\partial u}{\partial T}}\left[\frac{dm}{d\theta}_{iv}.(ho_{iv}-u) + \frac{dQ_w}{d\theta} - P.\frac{dV}{d\theta} - m.\frac{\partial u}{\partial f}.\frac{df}{d\theta}\right] \quad (4.9)$$

$$\frac{df}{d\theta} = \frac{1+f}{m}\left[(1+f).\left[\frac{f_{iv}}{1+f_{iv}}.\frac{dm}{d\theta}_{iv}\right] - f.\frac{dm}{d\theta}\right] \quad (4.10)$$

$$\frac{dm}{d\theta} = \frac{dm}{d\theta}_{iv} \quad (4.11)$$

o cylinder control volume during compression: inlet valve closed, exhaust valve closed, no combustion.

$$\frac{dm_{iv}}{d\theta} = \frac{dm_{ev}}{d\theta} = \frac{dm_f}{d\theta} = 0$$

substituting into the gas equations gives:-

$$\frac{dT}{d\theta} = \frac{1}{m.\frac{\partial u}{\partial T}}\left[\frac{dQ_w}{d\theta} - P.\frac{dV}{d\theta}\right] \qquad (4.12)$$

$$\frac{df}{d\theta} = 0$$

$$\frac{dm}{d\theta} = 0$$

o cylinder control volume during combustion: inlet valve closed, exhaust valve closed, combustion.

$$\frac{dm_{iv}}{d\theta} = \frac{dm_{ev}}{d\theta} = 0$$

substituting into the gas equations gives:-

$$\frac{dT}{d\theta} = \frac{1}{m.\frac{\partial u}{\partial T}}\left[\frac{dm_f}{d\theta}.(h_{for}-u) + \frac{dQ_w}{d\theta} - P.\frac{dV}{d\theta} - m.\frac{\partial u}{\partial f}.\frac{df}{d\theta}\right] \qquad (4.13)$$

$$\frac{df}{d\theta} = \frac{1+f}{m} . \frac{dm_f}{d\theta} \qquad (4.14)$$

$$\frac{dm}{d\theta} = \frac{dm_f}{d\theta} \qquad (4.15)$$

o cylinder control volume during power: inlet valve closed, exhaust valve closed, no combustion.

$$\frac{dm_{iv}}{d\theta} = \frac{dm_{ev}}{d\theta} = \frac{dm_f}{d\theta} = 0$$

substituting into the gas equations gives:-

$$\frac{dT}{d\theta} = \frac{1}{m.\frac{\partial u}{\partial T}}\left[\frac{dQ_w}{d\theta} - P.\frac{dV}{d\theta}\right] \qquad (4.16)$$

$$\frac{df}{d\theta} = 0$$

$$\frac{dm}{d\theta} = 0$$

o cylinder control volume during exhaust: inlet valve closed, exhaust valve open, no combustion.

$$\frac{dm_{iv}}{d\theta} = \frac{dm_f}{d\theta} = 0$$

Two models are used to represent the gas in this phase, one for the normal flow of gas through the exhaust valve, the other for reverse flow.

i) normal flow - from the cylinder control volume into the exhaust manifold control volume.

$$\frac{dT}{d\theta} = \frac{1}{m.\frac{\partial u}{\partial T}}\left[\frac{dm_{ev}}{d\theta}.(ho_{ev}-u) + \frac{dQ_w}{d\theta} - P.\frac{dV}{d\theta}\right] \qquad (4.17)$$

$$\frac{df}{d\theta} = 0$$

$$\frac{dm}{d\theta} = \frac{dm_{ev}}{d\theta} \qquad (4.18)$$

ii) reverse flow - from the exhaust manifold control volume into the cylinder control volume.

$$\frac{dT}{d\theta} = \frac{1}{m.\frac{\partial u}{\partial T}}\left[\frac{dm_{ev}}{d\theta}.(ho_{ev}-u) + \frac{dQ_w}{d\theta} - P.\frac{dV}{d\theta} - m.\frac{\partial u}{\partial f}.\frac{df}{d\theta}\right] \qquad (4.19)$$

$$\frac{df}{d\theta} = \frac{1+f}{m}\left[(1+f)\left[\frac{f_{ev}}{1+f_{ev}}\frac{dm_{ev}}{d\theta}\right] - f.\frac{dm}{d\theta}\right] \qquad (4.20)$$

$$\frac{dm}{d\theta} = \frac{dm}{d\theta}_{ev} \tag{4.21}$$

o inlet or exhaust manifold control volume: no combustion, constant volume.

$$\frac{dm_f}{d\theta} = \frac{dV}{d\theta} = 0$$

substituting into the gas equations gives:-

$$\frac{dT}{d\theta} = \frac{1}{m.\frac{\partial u}{\partial T}}\left[\sum \frac{dm}{d\theta}_i . ho_i + \sum \frac{dm}{d\theta}_o . ho_o - u.\frac{dm}{d\theta} + \sum \frac{dQ}{d\theta}_w - m.\frac{\partial u}{\partial f}.\frac{df}{d\theta}\right] \tag{4.22}$$

$$\frac{df}{d\theta} = \frac{1 + f}{m}\left[(1 + f).\frac{dm}{d\theta}_{fb} - f.\frac{dm}{d\theta}\right] \tag{4.23}$$

$$\frac{dm}{d\theta} = \sum \frac{dm}{d\theta}_i + \sum \frac{dm}{d\theta}_o \tag{4.24}$$

4.5 Control Volume Sub-Models

Evaluation of the right hand side of a control volume state equation requires the use of sub-models to represent the following features of engine behaviour:-

- o combustion
- o heat transfer
- o valve mass flow
- o turbocharger - turbine and compressor
- o cylinder volume and rate of change of volume
- o gas properties

The sub-models used for the experimental TL11 engine are

presented below.

4.5.1 Combustion Model

Since the speed response of an engine during a transient is determined primarily by its response to fueling, an accurate representation of the fuel combustion process is essential, and it is normally represented using a semi-empirical model. The model used, which was developed by Watson et al [4.6], predicts the rate at which fuel burns; this rate is then substituted into the gas control volume state equations, i.e the term:-

$$\text{o rate of fuel burning} \quad \frac{dm_f}{d\theta}$$

Figure 4.2 illustrates how the fuel burning rate (fbr) varies with crankshaft position, following an injection of fuel into a combustion chamber. This figure shows that combustion can be considered to occur in three stages. Firstly, there is the ignition delay period which starts immediately after entry of fuel into the chamber. There then follows rapid burning of the mixture prepared during the ignition delay period, and this is known appropriately as the "pre-mixed" burning phase. Finally, there is the slower diffusion phase of combustion, which is controlled by the availability of oxygen in the combustion chamber.

The model represents the fbr by separate expressions for the pre-mixed and diffusion modes of burning. Another expression (described in Section 4.5.1.1) is used to predict the ignition delay, during which the fbr is zero. At any instant the rate of fuel burning is equal to the sum of that due to pre-mixed and

diffusion burning, as is shown in Figure 4.3.

$$fbr(\tau) = fbr_p(\tau) + fbr_d(\tau) \qquad (4.25)$$

The quantities shown in the figure are non-dimensionalized with respect to the quantity of fuel injected into the cylinder and the nominal duration of burning. For convenience, both modes of burning are assumed to commence at ignition (τ=0) and are represented by the equations:-

- o pre-mixed mode of burning

$$fbr_p(\tau) = \beta.C_1.C_2.\tau^{C1-1}.(1 - \tau^{C_1})^{C2-1} \qquad (4.26)$$

- o diffusion mode of burning

$$fbr_d(\tau) = (1 - \beta).C_3.C_4.\tau^{C4-1}.\exp(-C_3.\tau^{C_4}) \qquad (4.27)$$

where τ is the crankshaft angle of rotation since ignition, non-dimensionalised by the nominal duration of burning - normally taken to be 125 degrees.

C_1-C_4 are constants for a particular engine operating condition.

β is referred to as the "mode of burning factor". It defines the proportion of the mass of fuel which is burnt in the pre-mixed mode of burning, to the total mass of fuel injected into the combustion chamber.

The fbr of the engine can be represented at any operating condition by Equations (4.25), (4.26) and (4.27), by choosing appropriate values for the parameters C1, C2, C3, C4 and β. The

relationships used to represent the dependency of these parameters on the operating regime of the engine, are obtained ideally using experimental data from the engine being modelled. This was not possible in this programme of research and instead the results obtained by Watson on a Leyland truck engine are used.

$$C_1 = 2.0 + 44.5(id)^{2.4}$$

$$C_2 = 5000.0$$

$$C_3 = \frac{2.5}{f^{0.644}} \qquad (4.28)$$

$$C_4 = 0.791(C_3)^{0.248}$$

$$\beta = 1.0 - 0.416\left[\frac{\omega_e}{id}\right]^{0.26}.f^{0.37}$$

where ωe is the engine speed

f is the trapped fuel/air ratio

id is the ignition delay radians

Finally, the rate of fuel burning for substitution into the gas state equations is calculated from the non-dimensional fbr Equation (4.25) using:-

$$\frac{dm_f}{d\theta} = \frac{m_f \cdot fbr(\tau)}{\Delta} \qquad (4.29)$$

where m_f is the mass of fuel injected

Δ is the nominal duration of combustion

The mass of fuel m_f, injected into the combustion chamber is calculated using the model of the fuel injector system which is described in Section 4.6.2. This model also calculates the crankshaft position at which fuel enters the combustion chamber θ_d,

from which the crankshaft position at which the fuel ignites θ_i can be calculated, once the ignition delay θ_{id} is known.

$$\theta_i = \theta_d + \theta_{id} \tag{4.30}$$

4.5.1.1 Ignition Delay Model

The ignition delay model is due to Wolfer [4.7], although the coefficients used are those which were suggested by Watson [4.8], for a high speed direct injection engine.

$$id = \frac{3.52}{P_m^{1.022}} \exp\left[\frac{2100}{T_m}\right] \quad (ms) \tag{4.31}$$

where P_m and T_m are the mean values of cylinder gas pressure (bar) and temperature (K) respectively, computed over the period from the injection of fuel into the combustion chamber to the instant of fuel ignition.

4.5.2 Heat Transfer Model

The heat transfer model is used to calculate the rate of heat transfer between the gas in a control volume and its surroundings; this value is then substituted into the gas state equations, i.e., the term:

- o rate of heat transfer $\frac{dQ_w}{d\theta}$

The influence which heat transfer has on a control volume, depends very much on which part of the engine system the control volume represents. Heat transfer in a cylinder, or an exhaust

manifold, makes a significant contribution to the overall energy balance, but is small in the inlet manifold.

The models used to represent heat transfer in a cylinder, and in a manifold are now discussed.

4.5.2.1 Heat Transfer in a Cylinder

The heat energy which is transferred between the gas in a cylinder and the combustion chamber walls is conducted away through engine components such as the piston, cylinder liner, inlet and exhaust valves etc to the water cooling system, and then to ambient surroundings. This heat transfer path is represented by the much simplified heat transfer model shown in Figure 4.4, in which heat energy is assumed to flow in one dimension only. Because thermodynamic equilibrium is assumed, it follows that the temperature of the gas in the control volume, and of its gas exposed surfaces is uniform. In Figure 4.4, R1 and R2 represent the effective thermal resistance between the cylinder gas and combustion chamber wall, and between the combustion chamber wall and ambient surroundings, respectively. The thermal capacitance of the heat transfer path R2 is represented by the capacitance C. Calculation of the thermal resistance R1, R2 and the thermal capacitance C is now considered.

Heat transfer between the gas in a cylinder and its gas exposed surfaces is due to forced convection and radiation (since loss by conduction is negligible), and can be described by the expression:-

$$\frac{dQ_w}{dt} = A.h(T - T_w) + A.\varepsilon.\sigma(T^4 - T_w^4) \qquad (4.32)$$

where h is the heat transfer coefficient

ε is the emissivity

σ is the Stephan-Boltzmann constant

It is generally agreed that during the induction, compression and exhaust strokes of engine operation, heat transfer is primarily convective in nature. For combustion however, there is considerable disagreement between investigators, as to the proportion of the total heat transferred by forced convection and radiation, some reporting that the contribution of radiation is negligible, and others, as high as 30% [4.9]. This disagreement is, in part, due to the difficulty of separately identifying the two contributions and to differences in engine type. If it is assumed that the contribution of radiation is negligible, then a simplified heat transfer model may be used.

$$\frac{dQ_w}{dt} = A.h(T - T_w) \qquad (4.33)$$

This simplified model was adopted to represent heat transfer in a cylinder. With this model, it follows that the thermal resistance R_1, is the reciprocal of the product of the heat transfer coefficient and the combustion chamber wall area.

$$R_1 = \frac{1}{A.h} \qquad (4.34)$$

The major difficulty in calculating heat transfer in a cylinder lies in estimating the heat transfer coefficient h, which depends primarily on the flow conditions in the vicinity of the

cylinder walls. The movement of gas inside a cylinder is so complex, with flow entering and leaving through the valves, the piston moving up and down at high speed and combustion taking place, that a mathematical description of the flow is quite impracticable. Consequently, the heat transfer coefficient has to be estimated using a semi-empirical model, which represents the effect of the principal variables known to influence heat transfer in a cylinder, such as mean piston speed. The relationship used is the one due to Hohenberg [4.10], who conducted a series of experiments on direct injection truck sized diesel engines. Hohenberg made the assumption that heat transfer in a cylinder is entirely convective in nature and from his investigations proposed the following expression to predict the heat transfer coefficient.

$$h = \frac{C_1 . P^{0.8} (\overline{V}_p + C_2)^{0.8}}{V^{0.06} . T^{0.4}} \qquad (4.35)$$

where $\overline{V}_p$ is the mean piston speed

In this equation C_1 and C_2 are constants for a particular engine and ideally should be determined from heat transfer measurements on the engine being modelled. In practice this is rarely possible and it is usual to use the values recommended by Hohenberg.

$$C_1 = 130.0 \qquad (4.36)$$

$$C_2 = 1.4$$

The surface area of the cylinder exposed to the gas, which is to be substituted into Equation (4.33), is calculated using the

equation:

$$A = A_1 + \pi d\left(crl + r.(1 - \cos\theta) - \sqrt{(crl^2 - r^2.\sin^2\theta)}\right) \quad (4.37)$$

where d is the cylinder diameter

crl is the connecting rod length

r is the crank throw radius

The area A_1 includes the area of the combustion chamber and piston land. Since the velocity and temperature of the gas in the vicinity of the piston land is lower than that in the main combustion chamber, the model (which assumes thermodynamic equilibrium), overestimates the heat flow from the cylinder. To correct for this, Hohenberg, somewhat arbitrarily, proposed using a value for the land area which is equal to the actual area of the piston land raised to the power 0.3.

The thermal resistance between the combustion chamber wall and ambient is calculated using a one dimensional heat transfer path, through the engine components, coolant, and coolant to ambient interface. During transient operation of the engine, the temperature of the combustion chamber wall will change in response to the operating condition of the engine, but because of the large thermal mass of the heat transfer path, the change will be very gradual. The thermal capacitance of the heat transfer path is represented by the capacitor shown in Figure 4.4, and its value is chosen to give an appropriate time constant.

The rate of change of the difference in temperature between the combustion chamber wall and ambient is calculated using:-

$$\frac{d(\Delta T)}{dt} = \frac{1}{C}\frac{dQ_w}{dt} - \frac{\Delta T}{C.R2} \qquad (4.38)$$

where $\Delta T = T_w - T_{ambient}$

4.5.2.2 Heat Transfer in a Manifold

Figure 4.4 shows the simple one dimensional heat flow model used to represent heat transfer between the manifold gas and ambient surroundings, and is identical to that used to represent heat transfer from a cylinder. However, in this case, R1 and R2 now represent the effective thermal resistances of the manifold gas to the internal manifold wall surface, and the internal manifold wall surface to ambient surroundings, respectively. Heat transfer between the gas in the manifold, and the manifold internal wall surface (R1) is assumed to be convective in nature, and is represented by Equation (4.33). The heat transfer coefficient h, is given a constant value calculated from the theory of heat transfer in pipes.

The thermal resistance R2 represents conduction through the manifold wall and heat transfer from the external surface of the manifold to the surrounding air. The capacitance shown in Figure 4.4 represents the thermal capacitance of the manifold casting. The rate of change of the difference in temperature between the internal wall of the manifold and ambient surroundings, is calculated using Equation (4.38).

4.5.3 Valve Model

The control volumes representing the inlet and exhaust manifolds are connected to the control volumes representing the engine cylinders, by a flow of gas through the valves. The valve model first determines the direction of flow through a valve, and then calculates the mass flow rate, for substitution into the control volume state equations: i.e., the terms:

- rate of mass flow $\frac{dm_{iv}}{d\theta}$ - inlet valve flow

  $\frac{dm_{ev}}{d\theta}$ - exhaust valve flow

The valve is represented by an orifice having an equivalent flow area C_dA, which, of course, varies cyclically as the valve opens and closes during the course of engine operation. It is assumed that the complex unsteady flow of gas through a valve can be represented on a quasi-steady basis, by one dimensional compressible steady flow theory. Upstream of the orifice throat it is assumed that the flow is isentropic, while downstream, a constant static pressure model is used, with no pressure recovery of the gas velocity.

Gas flow through an orifice throat is either subsonic or supersonic, depending upon the state of the gas in the control volumes on either side of the orifice. The pressure ratio at which the transition between the two types of flow occurs, is known as the critical pressure ratio. The pressure ratio is defined as:-

$$P_r = \frac{P_u}{P_d} \tag{4.39}$$

where P_u is the pressure in the upstream volume

P_d is the pressure in the downstream volume

and the critical pressure ratio is calculated using:-

$$P_{crit} = \left(\frac{\gamma + 1}{2}\right)^{\gamma/(\gamma-1)} \tag{4.40}$$

where γ is the ratio of specific heats

If the pressure ratio is less than the critical value, the flow of gas is subsonic, and calculated using:

$$\frac{dm}{dt} = C_d \cdot A \cdot P_u \left[\frac{2.\gamma}{R.T_u.(\gamma-1)}\left[\left(\frac{1}{P_r}\right)^{2/\gamma} - \left(\frac{1}{P_r}\right)^{(\gamma+1)/\gamma}\right]\right]^{1/2} \tag{4.41}$$

otherwise the flow is supersonic ($P_r > P_{crit}$) and is calculated using:

$$\frac{dm}{dt} = C_d \cdot A \cdot P_u \left[\frac{\gamma}{R.T_u}\left(\frac{2}{\gamma + 1}\right)^{(\gamma+1)/(\gamma-1)}\right]^{1/2} \tag{4.42}$$

4.5.3.1 Effective Flow Area of a Valve

Since the flow model ignores secondary flow effects such as friction, it overestimates the mass flow rate through the valve. To compensate for this, albeit somewhat empirically, a correcting factor is applied, referred to as the discharge coefficient, C_d. Investigations [4.11], have shown that for a particular valve and port combination, the value of the discharge coefficient varies primarily with the ratio of valve lift to valve diameter, and to a lesser extent, with the pressure ratio across the valve. Because

the effect of pressure ratio is small, it is usually ignored, and the effective flow area of the valve C_dA, used directly in the engine model. Ideally, the effective flow area should be obtained from steady flow tests on the valve being modelled, although for practical reasons, it was not possible to do this for the valves used in the TL11 engine. Consequently, the effective flow areas of the inlet and exhaust valves had to be calculated from a knowledge of the valve lift profiles, valve geometry and discharge coefficients, which were estimated from data given in reference [4.12].

4.5.4 Turbocharger Model

The engine inlet manifold is supplied with air from the turbocharger compressor and the two exhaust manifolds eject hot exhaust gas into the turbocharger turbine unit, thereby developing the torque required to drive the compressor. The compressor is a conventional centrifugal unit, but the turbine is a variable geometry unit, which can be controlled by the position of the turbine nozzle area actuator. The model representing the behaviour of the compressor and turbine units, evaluates the rate of mass flow through the compressor and turbine units for substitution into the control volume state equations. In addition, the model calculates the torque developed by the turbine, and the torque required to drive the compressor, which are used to calculate the angular acceleration of the turbocharger shaft.

The assumption is made that the complex non-steady flow of gas through the compressor and turbine units, can be represented as

though it were quasi-steady. Hence, the flow and efficiency characteristics of the compressor and turbine units are represented by their steady flow performance maps, which are normally obtained from experimental measurements.

The compressor and turbine models are now discussed, followed by a discussion of the model used to represent the dynamic behaviour of the turbocharger.

4.5.4.1 Compressor Model

The compressor is modelled by its steady flow performance map, which is shown in Figure 4.5. Given the pressure of the gas upstream of the compressor (i.e. ambient), and downstream of the compressor (i.e. inlet manifold), and the rotational speed of the turbocharger shaft ω_{tc}, then the mass flow rate of gas through the compressor, and compressor efficiency η_c, can be found by interpolation on the map. Compressor isentropic efficiency is defined as the ratio of isentropic work to actual work, from which the temperature of the flow of gas from the compressor unit into the inlet manifold is obtained using:

$$T_d = T_u + \frac{T_u}{\eta_c}\left[\left[\frac{P_d}{P_u}\right]^{(\gamma-1)/\gamma} - 1\right] \qquad (4.43)$$

Finally the power and shaft torque required to drive the compressor are calculated, using:

$$P_c = \frac{dm_c}{dt}.\Delta h \qquad (4.44)$$

$$T_c = \frac{P_c}{\omega_{tc}} \qquad (4.45)$$

where Δh is the change in enthalpy of the gas.

4.5.4.2 Turbine Model

The turbine is modelled by assigning an imaginary turbine to each exhaust manifold. Both turbines are constrained to rotate at the same speed, and their mass flow performances are scaled so that their combined mass flow is equal to that of the real turbine. Originally, it was intended to represent the mass flow and efficiency performance of the turbine, over its operational range, by its steady flow performance map. Given the state of the gas upstream and downstream of the turbine, the rotational speed of the turbocharger shaft and turbine nozzle restriction, the mass flow rate of gas through the turbine, and turbine efficiency, would then have been found by interpolation on the performance map. Unfortunately, the manufacturers of the turbine were unwilling to supply the performance map, and consequently an alternative model had to be used.

The turbine was modelled very simply using the swallowing curve shown in Figure 4.6. This was measured by Roberts [4.13], with the engine operating on its limiting torque curve and with the turbine restriction fully open. The use of the characteristic implies that the turbine performance is independent of the speed parameter, which is a simplification. The effect of turbine restriction on the mass flow behaviour of the turbine was modelled using a scale factor which reduces turbine mass flow linearly with

turbine restriction. This is a reasonable approximation to make, since Roberts has shown that the relationship between mass flow and turbine restriction is approximately linear. Thus, given the pressure ratio across the turbine, the temperature of the gas at the turbine inlet and turbine restriction, then the rate of mass flow through the turbine can be found by interpolating the characteristic.

Turbine isentropic efficiency is defined as the ratio of actual turbine work to isentropic work, from which the temperature of the flow of gas at the turbine exit is calculated using:

$$T_d = T_u - \eta_t . T_u \left[1 - \left(\frac{P_u}{P_d} \right)^{(1-\gamma)/\gamma} \right] \qquad (4.46)$$

The torque developed by each imaginary turbine is given by:-

$$T_i = \frac{dm_i}{dt} \frac{\Delta h_i}{\omega tc} \qquad (4.47)$$

Finally, the total turbine torque is obtained by summing the torque contributions of both imaginary turbines.

Although simplifications have had to be made in modelling the turbine, it must be stressed that they have little effect on the overall model execution time. Clearly, the amount of computation required for the simple turbine model is less than that required for a more detailed model, but since the turbine model represents only one element in an exhaust manifold model, and an exhaust manifold model constitutes only one element in the total engine model, the simplifications made cannot significantly affect the model run time results which are presented in Chapter 9.

4.5.4.3 Turbocharger Acceleration

The angular acceleration of the turbocharger shaft is calculated using the inertia equation:-

$$\frac{d\omega_{tc}}{dt} = \frac{T_t - T_c}{J_{tc}} \qquad (4.48)$$

where T_t is the torque developed by the turbine

T_c is the torque required to drive the compressor

J_{tc} is the effective inertia of the rotating parts of the turbocharger

4.5.5 Cylinder Volume and Rate of Change of Volume

The volume of gas in the engine cylinders is required for substitution into the perfect gas equation (4.1), and the rate of change of volume of gas in the cylinder, is required for substitution into the control volume state equation (4.2). The equation for calculating the volume of gas in the cylinder is derived from the action of the crank and connecting rod mechanism, as shown in Figure 4.7 and is given by.

$$V = \frac{\pi d^2}{4}\left[crl + r(1 - \cos\theta) - \sqrt{(crl^2 - r^2.\sin^2\theta)} + \frac{2.r}{C_r - 1}\right] \qquad (4.49)$$

where C_r is the compression ratio

Differentiating with respect to crankshaft angle gives

$$\frac{dV}{d\theta} = \frac{\pi d^2}{4}\left[r.\sin\theta + \frac{r^2.\sin\theta.\cos\theta}{\sqrt{(crl^2 - r^2.\sin^2\theta)}}\right] \qquad (4.50)$$

4.5.6 Gas Property Model

The gas property model calculates the values of the following quantities for use by the engine model:

- o u - specific internal energy
- o $\frac{\partial u}{\partial T}$ - partial derivative of specific internal energy with respect to temperature
- o $\frac{\partial u}{\partial f}$ - partial derivative of specific internal energy with respect to gas composition
- o R - gas constant
- o q - ratio of specific heats
- o ho - specific stagnation enthalpy

Although the specific internal energy u, is a function of gas temperature, pressure, and composition, its dependence on pressure is small, and is ignored in the modelling. The gas constant R depends upon gas composition only, then:

$$u = f(T,f) \qquad (4.51)$$

$$R = f(f)$$

Tabulated values of specific internal energy and the gas constant are available for the products of combustion and air, and it is usual when modelling the gas property behaviour to use polynomial expressions which have been derived from the data. The formulae used in the engine model for lean fuel air ratio ($f/f_s < 1$), are given below. These relationships are taken from Krieger et al [4.14] and are based on the data of Newall et al [4.15] for the combustion products of diesel fuel (C_nH_{2n}) and air.

$$u = \frac{K1(T) - K2(T).f}{1 + f} \quad \text{per unit mass of combustion products} \qquad (4.52)$$

where $K1(T) = 692.0T + 39.17 \times 10^{-3}T^2 + 52.9 \times 10^{-6}T^3$
$-228.62 \times 10^{-10}T^4 + 277.58 \times 10^{-14}T^5$

$K2(T) = 4.5109 \times 10^{7} - 843.195T - 1.4053T^2$
$+ 3.1849 \times 10^{-4}T^3 - 2.9624 \times 10^{-8}T^4$

$$R = \frac{287.0 + 295.86*f}{1 + f} \quad \text{per unit mass of combustion products} \qquad (4.53)$$

$$\gamma = 1.0 + \frac{R}{\frac{\partial u}{\partial T}} \qquad (4.54)$$

$$ho = u + R*T \qquad (4.55)$$

$\frac{\partial u}{\partial f}$ and $\frac{\partial u}{\partial T}$ are obtained from Equation (4.52)

$$\frac{\partial u}{\partial f} = \frac{-K1(T) - K2(T)}{(1 + f)^2} \qquad (4.56)$$

$$\text{and } \frac{\partial u}{\partial T} = \frac{K3(T) - K4(T)*f}{1 + f} \qquad (4.57)$$

where $K3(T) = 692.0 + 7.834 \times 10^{-2}T + 1.587 \times 10^{-4}T^2$
$- 9.1448 \times 10^{-8}T^3 + 1.3879 \times 10^{-11}T^4$

$K4(T) = -843.195 - 2.8107T + 9.5547 \times 10^{-4}T^2$
$- 1.185 \times 10^{-7}T^3$

4.6 Description of the Dynamic Engine Model

The gas state equations and the sub-models which have just been described, represent the behaviour of the gas in the engine.

Solving these equations using constant values for engine speed, fueling, fuel injection timing and turbine nozzle restriction will yield the cyclic, steady state behaviour of the gas in the engine at the assumed operating condition. Previous attempts to calculate the dynamic response of engines to time varying inputs using filling and emptying models, have been very restricted by the inordinately long time taken to compute the model equation. The use of parallel processing will reduce the model execution time by a significant factor and should make the use of filling and emptying models a much more attractive proposition in dynamic response studies.

Representation of the dynamic response of an engine requires additional models and these are now considered.

4.6.1 Control Actuator Model

The experimental TL11 diesel engine has three control inputs which allow control to be exercised over the quantity of fuel injected into the engine, its timing, and the turbine nozzle restriction. All three control inputs are actuated by hydraulic actuators which are operated in closed loop, as is described in Chapter 3.

The mathematical models used to represent the actuators were derived using a system identification method. A series of experiments were performed on the fuel rack and turbine restriction actuators, to record their response to step and pseudo random binary sequence (prbs) input disturbances. The responses were analysed (using the least squares technique) to obtain z-transform models,

which represent the dynamic behaviour of the actuators. An inverse transformation was then applied to the z-transform models to obtain continuous time domain models for use with the engine model. The experimental programme, and analysis of the results, are described in Appendix A2.

As was to be expected from physical considerations, it was found that the actuator dynamics are dominantly second order in nature, with a velocity limit which is determined by the maximum flow of hydraulic fluid. Thus the dynamic response of the actuators can be represented by:

for $\dot{x}_a < \dot{x}_{slew}$

$$\frac{x_{a(s)}}{x_{d(s)}} = \frac{K.\omega_n^2}{s^2 + 2.\zeta.\omega_n.s + \omega_n^2} \qquad (4.58)$$

for $\dot{x}_a > \dot{x}_{slew}$

$$\dot{x}_a = \dot{x}_{slew}$$

where x_a is the actuator position

x_d is the demanded actuator position

Values for the model coefficients and the slew rate limit are given in Table 4.1.

It was not possible to identify a model of the fuel timing actuator because the actuator had been removed from the engine for repair. Consequently, the coefficients used in this model had to be calculated from its physical characteristics.

4.6.2 Fuel Delivery System

The fuel delivery system consists of the fuel pump, fuel delivery pipes and the fuel injectors, and is modelled in order to provide the inputs which are required by the combustion model. The combustion model requires information on the crankshaft position at which fuel enters the combustion chamber θ_d, and the quantity of fuel injected m_f. Since the combustion model does not require the rate of change of fuel mass entering the combustion chamber, the fuel delivery system can be modelled very simply by the steady flow performance characteristic of the fuel pump and a transport delay which represents the time taken for fuel to travel the distance between the fuel pump and the fuel injectors.

The steady flow characteristic of the fuel pump used (‡) is given in Figure 4.8, and shows that fuel delivery depends upon rack position and engine speed.

$$m_f = f(z,\omega_e) \tag{4.59}$$

The quantity of fuel injected m_f is obtained by interpolating the characteristic using the value of fuel rack position and engine speed at the instant that fuel enters the combustion chamber θ_d.

(‡) The fuel pump delivery characteristic shown was measured when the fuel pump rack was actuated by a pneumatic system. Subsequently the actuation system was changed (to a hydraulic system) and it is not known how the rack position of the new hydraulic system corresponds to the rack position of the old pneumatic system, shown in Figure 4.8.

This assumes that the fuel rack position and engine speed do not change during the fuel pump plunger stroke, which is a reasonable approximation to make, since the fuel injection period is extremely short.

The crankshaft position at which fuel enters the combustion chamber θ_d, is calculated from the crankshaft position at which the fuel pump operates (static timing θ_s), and the fuel transport delay θ_t.

$$\theta_d = \theta_s + \theta_t \tag{4.60}$$

Static timing θ_s, is determined from the position of the fuel timing actuator. The transport delay θ_t, is approximately equal to the time it takes the fuel pressure wave (caused by the fuel pump pressurizing a small quantity of fuel), to travel the length of the delivery pipe. The pressure wave propagates at the speed of sound, and therefore the transport delay (in radians) is calculated using:-

$$\theta_t = \frac{l.\omega_e}{C} \tag{4.61}$$

where l is the length of the fuel delivery pipe

C is the speed of sound

4.6.3 Engine Acceleration

The angular acceleration of the engine is calculated using the inertia equation:

$$\frac{d\omega_e}{dt} = \frac{T_b - T_d}{J_e + J_d} \tag{4.62}$$

where J_e is the effective inertia of the engine

J_d is the effective inertia of the dynamometer

The brake torque T_b is obtained by subtracting the engine friction torque loss T_f, from the indicated torque T_i developed by the gas in the engine cylinders.

$$T_b = T_i - T_f \tag{4.63}$$

The indicated torque is obtained by summing the individual torque contributions from all six engine cylinders.

4.6.3.1 Engine Friction

An expression is required to calculate the friction losses in the engine, for use with Equation (4.63). Engine losses can be divided into two categories. Firstly, there are the genuine friction losses, which occur between sliding surfaces in bearings and in the valve gear etc, and secondly there is the power, also effectively lost, which is required to drive engine auxiliaries, such as the water pump, oil pump, cooling fan etc, without which the engine could not operate. When modelling the losses, it is normal to combine both categories and refer to them as engine friction.

Engine losses are normally measured during an experiment in which the power required to rotate the engine (usually using an electric motor), is measured. The results obtained indicate the mean level of engine friction. Millington and Hartles [4.16] and Chen and Flynn [4.17] proposed simple models to represent the friction loss results obtained from motoring tests, for a naturally aspirated and a turbocharged engine respectively. Since the TL11 engine is turbocharged, the Chen and Flynn model was adopted. This model represents the losses by a constant term, a term proportional

to mean piston speed, and a term proportional to peak cylinder pressure:-

$$fmep = C_1 + C_2.\overline{V}_p + C_3.P_{max} \qquad (4.64)$$

where fmep is friction mean effective pressure (bar)

Pmax is the peak cylinder pressure

$\overline{V}_p$ is the mean piston speed

Ideally, the coefficients used in the model (C_1, C_2 and C_3) should be obtained from motoring experiments on the engine being modelled. This was not possible for the TL11 engine and the coefficients used to represent the frictional losses were taken from results obtain by Chen et al [4.17].

4.7 Summary

The filling and emptying engine model used to represent the dynamic response of the experimental TL11 engine has been described. The basis of the model is to represent the engine cylinders and manifolds using thermodynamic control volumes which are interconnected by the transfer of mass and energy between volumes. The behaviour of the gas in the individual control volumes is represented by state equations, representing the rates of change of gas temperature, fuel/air ratio and mass. The state equations used for a cylinder and manifold control volume during the different phases of the engine power cycle have been presented.

Evaluation of the state equations requires the use of sub-models which represent various physical processes within the engine. Combustion in the engine cylinders is represented using the Watson

et al combustion correlation [4.6]. Heat transfer in the engine cylinders and manifolds is assumed to be entirely convective in nature, with the convective heat transfer coefficient being taken from Hohenberg [4.10] for the cylinders and a relationship for pipe flow being used in the manifolds. The gas properties are evaluated using polynomial expressions which have been curve fitted to combustion product data [4.14]. One dimensional, compressible steady flow theory is used to calculate the mass flow of gas through the engine valves which are modelled as an orifice of an equivalent flow area. The turbocharger compressor is represented by its steady flow performance map and the turbine by a swallowing curve.

The basic filling and emptying model of the gas in the engine cylinders and manifolds is extended to represent the dynamics of the control actuators, the fuel injection system and the engine-load speed dynamics, thereby enabling speed transients to be simulated. The control actuators are represented by second order linear differential equations with a slew rate limit, the fuel pump by its steady flow delivery characteristic plus a transport delay. The speed dynamics of the engine and load system are represented by the inertia equation.

In total, forty three state equations are used to represent the dynamic response of the experimental TL11 engine.

4.8 References

4.1 G L Borman.
Mathematical Simulation of Internal Combustion Engine Processes.
1964, PhD Thesis, University of Wisconsin.

4.2 K J McAulay, W Tang, S Chen, G Borman, P Myers, O Uyehara.
Development and Evaluation of the Simulation of the Compression Ignition Engine.
1965, SAE 650451.

4.3 E E Streit, G L Borman.
Mathematical Simulation of a large Turbocharged Two-stroke Diesel Engine.
1971, SAE 710176.

4.4 M Marzouk.
Simulation of Turbocharged Diesel Engines Under Transient Conditions.
1976, PhD Thesis, University of London (Imperial College).

4.5 Research in Internal Combustion Engineering in the U.K. Universities and Polytechnics. A Symposium organized by the Universities Internal Combustion Engine Group.
17/18 April 1980, Kings College London.

4.6 N Watson, M Marzouk, A D Pilley.
A Combustion Correlation for Diesel Engine Simulation.
February 1980, SAE 800029.

4.7 H Wolfer.
Ignition Lag in Diesel Engine.
VDI Forschungsheft No.392 1938.
Translation by Royal Aircraft Establishment, Farnborough Library No. 358.
August 1959, UDC No. 621-436 047.

4.8 N Watson.
E17 Combustion and Gas properties.
September 1979, Department of Mechanical Engineering, Imperial College, London.

4.9 N Watson, M S Janota.
Turbocharging the Internal Combustion Engine.
1982, Macmillan Press, London.

4.10 G F Hohenberg.
Advanced Approaches for Heat Transfer Calculations.
1979, SAE 790825.

4.11 W J D Annard, G E Roe.
Gas Flow in the Internal Combustion Engine.
1974, Foulis.

4.12 B R Garnish.
The Estimation of Discharge Coefficients for Poppet Valves.
Dec 1967, English Electric Company, Whetstone.

4.13 E W Roberts.
Variable Geometry Turbocharging Optimisation and Control.
1984, PhD Thesis, University of Bath.

4.14 R B Krieger, G L Borman.
The Computation of Apparent Heat Release for Internal Combustion Engines.
1966, ASME 66-WA/DGP-4.

4.15 H K Newall, E S Starkman.
Thermodynamic Properties of Octane and Air for Engine Performance Calculations.
1964, SAE Progress in Technology Series Vol 7.

4.16 B W Millington, E R Hartles.
Frictional Losses in Diesel Engines.
1968, SAE 680590.

4.17 S K Chen, P Flynn.
Development of a Compression Ignition Research Engine.
1965, SAE 650733.

4.9 Tables

Actuator	Gain K	Damping Ratio ζ	Undamped Natural Frequency (rad/s) ω_n	Slew rate (m/sec)
Fuel Rack Actuator	0.98	0.55	174	0.1
Turbine Nozzle Actuator	1.05	0.76	142	0.8
Fuel Injection Timing Actuator	1. 0	0.7	125	0.2

Table 4.1 Engine Control Actuator Response.

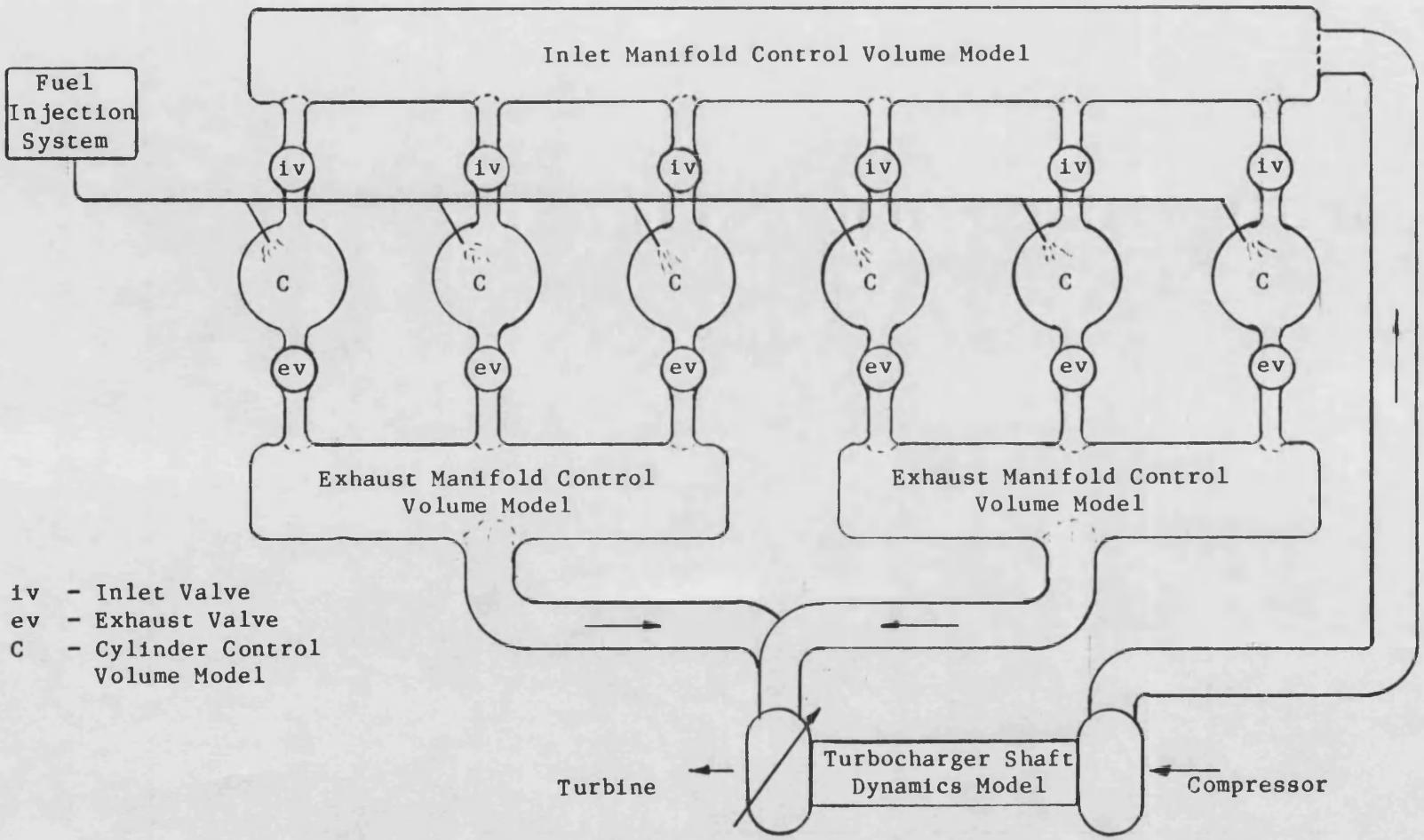


Figure 4.1 Filling and Emptying Model Representation of the TL11 Engine.

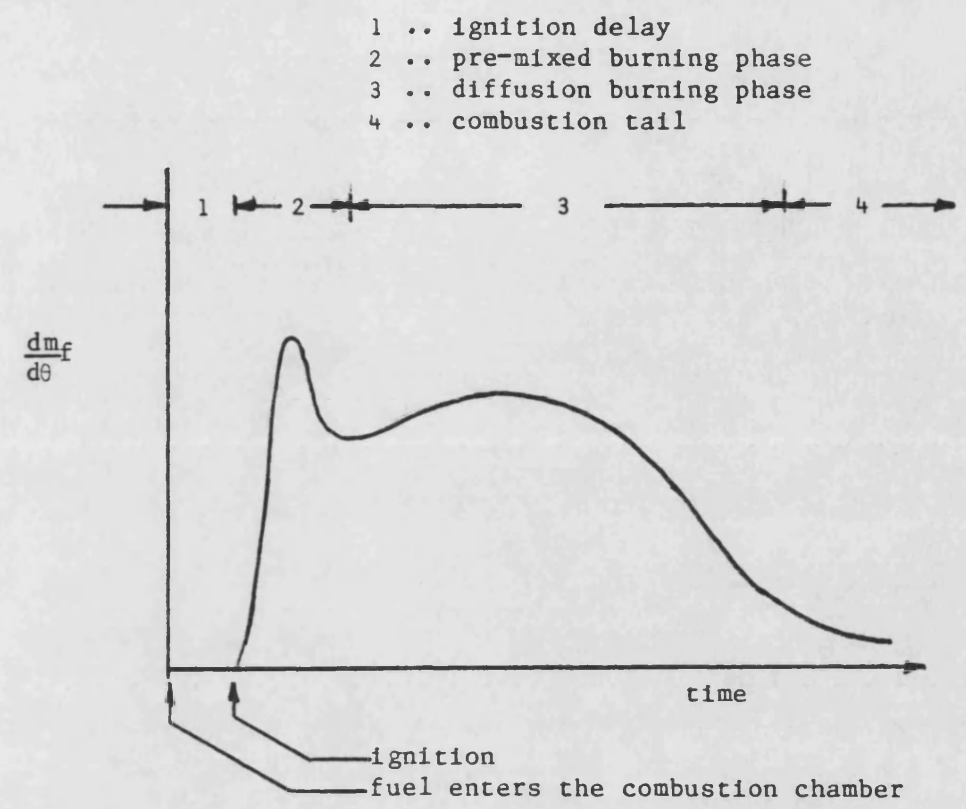
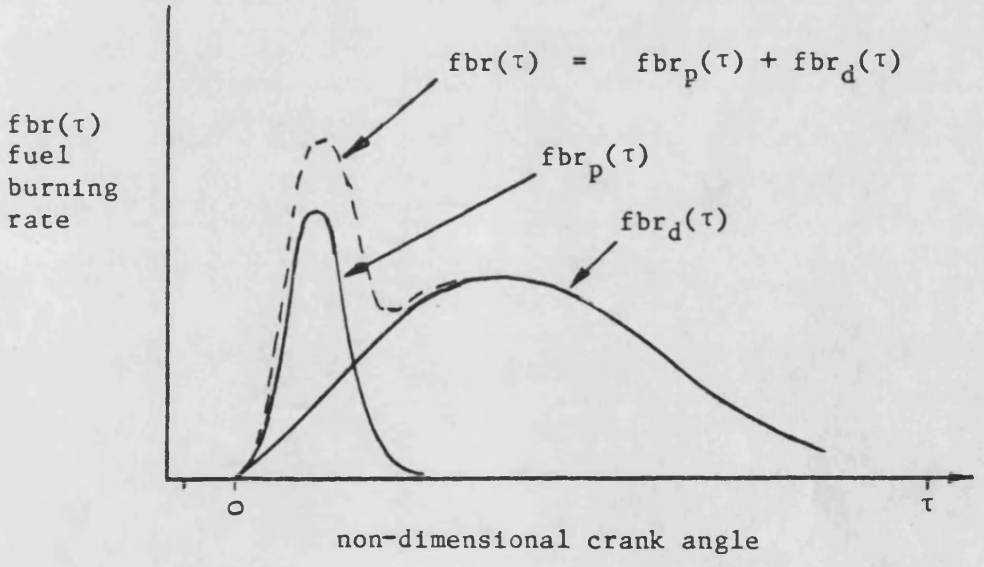


Figure 4.2 Phases of the Combustion Process.



fbr$_p$(τ) .. pre-mixed fuel burning rate
fbr$_d$(τ) .. diffusion fuel burning rate

Figure 4.3 Combustion model.

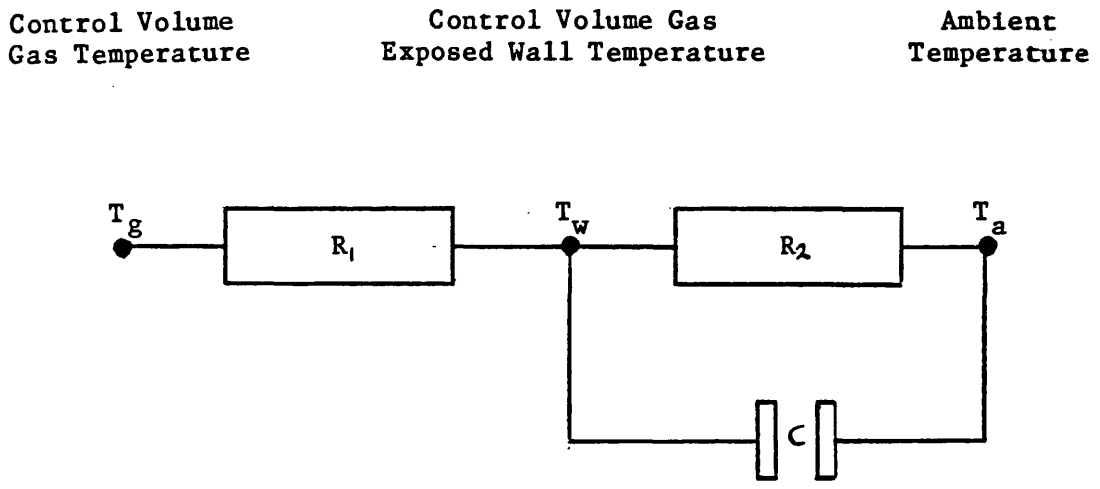


Figure 4.4 Control Volume Heat Transfer Model.

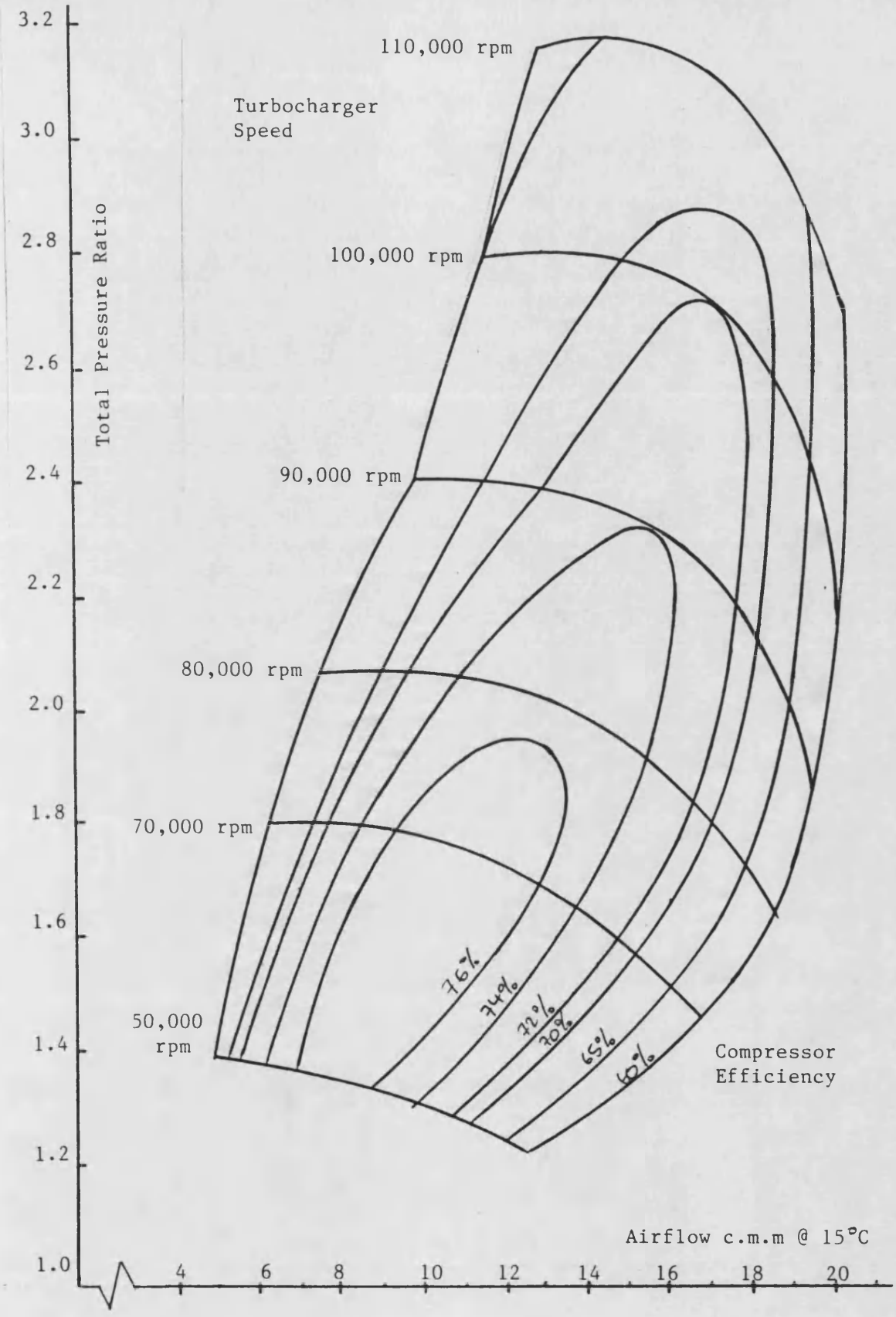


Figure 4.5 Compressor Characteristic.

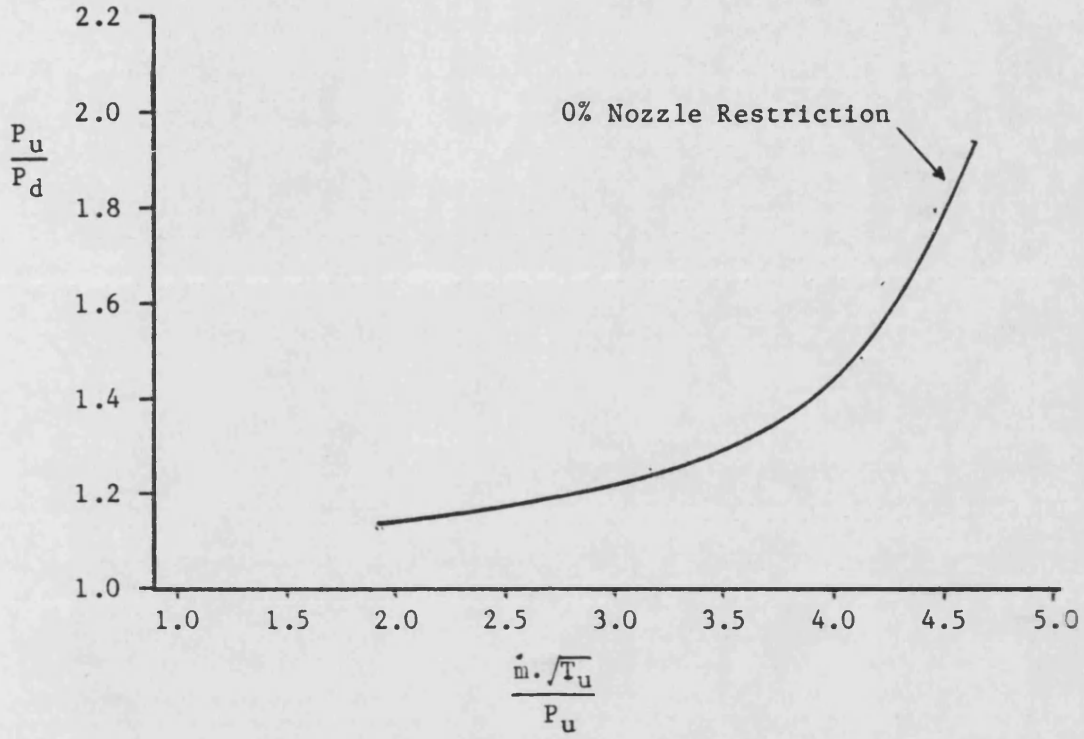


Figure 4.6 Turbine Swallowing Characteristic.

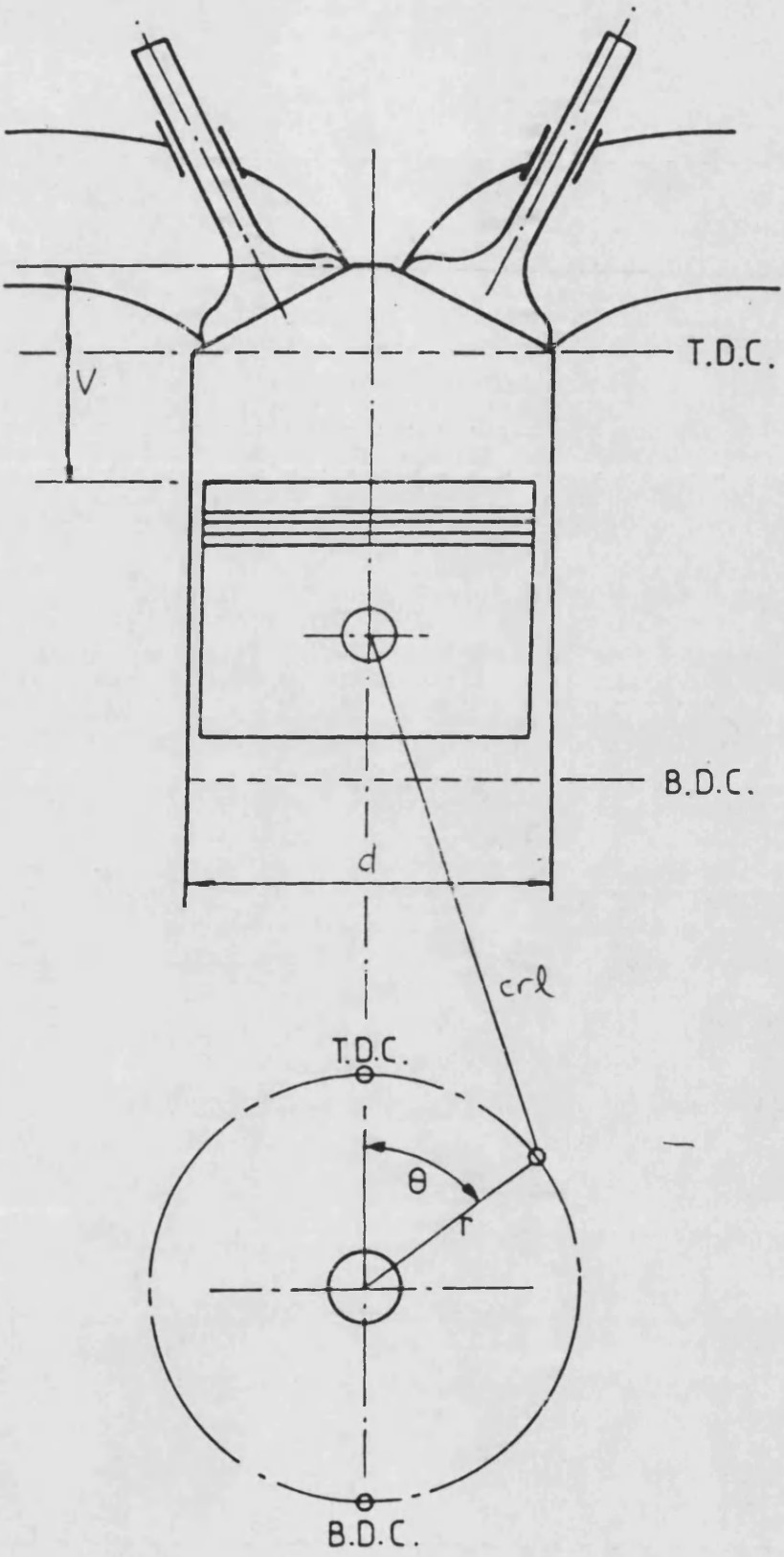


Figure 4.7 Crank and Connecting Rod Mechanism.

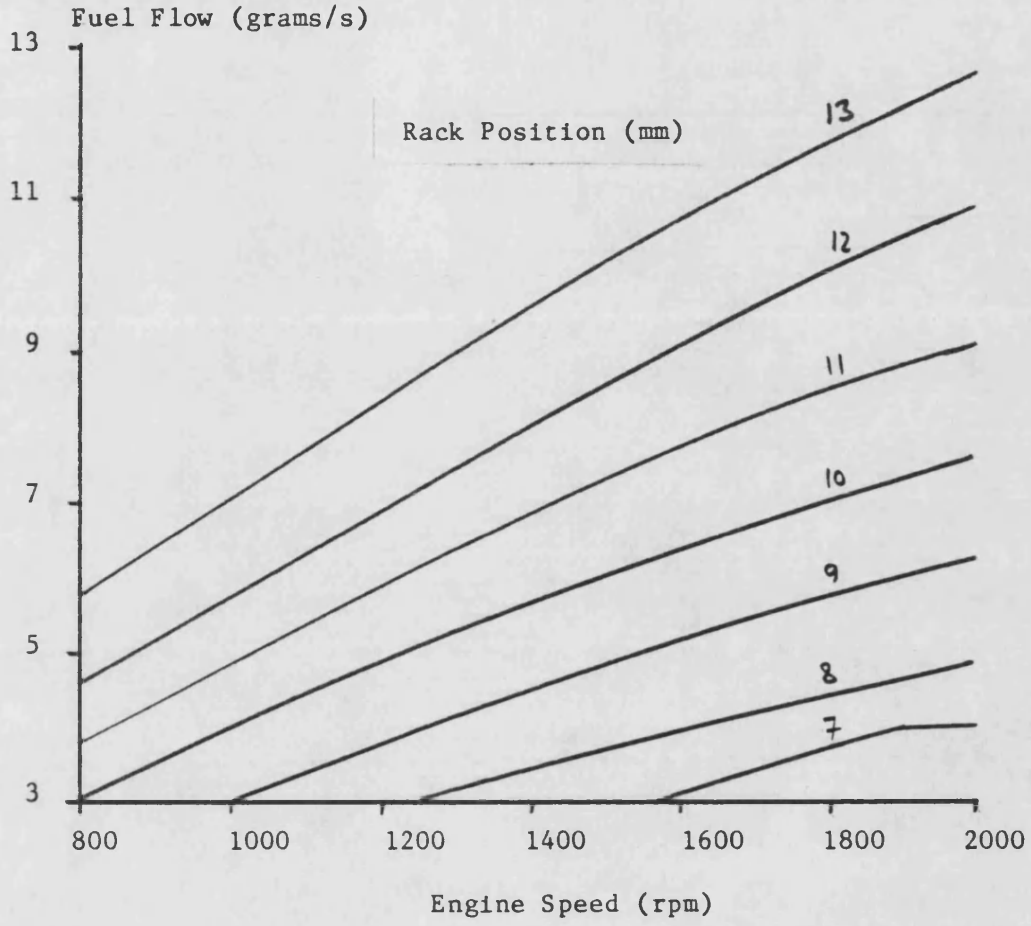


Figure 4.8 Fuel Pump Delivery Characteristic.

CHAPTER 5

5.1 The Diesel Engine Simulator

This chapter describes how the filling and emptying diesel engine model can be divided into a number of tasks which can be computed concurrently, with a resultant reduction in computational time (Section 5.2). The chapter also describes the method used to solve the model equations (Section 5.3), the technique used to schedule the model tasks amongst the physical processing resources of the computer system (Section 5.4), and how "look up" tables can be used to further improve the execution speed of the engine model (Section 5.5). Although the primary objective of this programme of research was to evaluate the feasibility of solving the diesel engine filling and emptying model in parallel, it was also considered highly desirable to produce an engine model which is of practical value. This required the development of a "diesel engine simulator" based, of course, on solving the filling and emptying diesel engine model in parallel. The general requirements for the simulator are described in Section 5.6.

5.2 Division of the Filling and Emptying Engine Model for Computing Concurrently

In order to compute the engine model in parallel, concurrency must be identified in its operation. This is vitally important, since it determines the time spent by the processors communicating

with one another, the computational balance achieved, as well as the program structure. Most of the effort spent in computing the filling and emptying model is expended in evaluating and integrating the control volume state equations. Fortunately, concurrent operation can be readily identified at this level because of the manner in which the engine operates. Each cylinder in the engine operates relatively independently of the others, with the crankshaft, fuel shot and valve flows being the only external influences affecting its behaviour. The engine manifolds also operate relatively independently of the rest of the engine, being influenced only by the flow of gas entering and leaving the volume. These features of engine operation, have been used as the basis for dividing the engine model into a number of tasks for computing concurrently, with the division between tasks being made along thermodynamic control volume boundaries.

The engine model was divided along control volume boundaries for several important reasons. Firstly, the control volumes are relatively independent, which minimises the need for them to exchange information. Secondly, the thermodynamic control volume is the fundamental computational partition of the filling and emptying model. Thirdly, the division results in a neat program structure, since only two fundamental tasks have to be developed, one representing a manifold control volume, and the other a cylinder. Furthermore, it has the considerable practical advantage that quite different engine configurations can be simulated with minimal change. For example, to simulate a single cylinder engine requires one cylinder control volume and two manifold control volume tasks to

be loaded onto the computer system, and to simulate a six cylinder engine requires the addition of five more (identical) cylinder control volume tasks, and possibly one more manifold control volume task.

Additional tasks are required to represent the dynamics of the control actuators and engine shafts (crankshaft and turbocharger shaft), but these are trivial in complexity and computational requirement, compared to the control volume tasks. Finally, a supervisor task is required to manage the engine model solution procedure, and to exchange information between the various engine model tasks.

5.3 Method of Engine Model Solution

The filling and emptying engine model is solved for a crankshaft angle $\theta+\Delta\theta$ from a knowledge of the state of the engine at a crankshaft angle θ. The control volume state equations which represent the rates of change of gas temperature (4.2), gas fuel-air ratio (4.3) and gas mass (4.4), in all the control volumes are integrated numerically and this procedure advances the gas solution in all the thermodynamic control volumes by the increment of crankshaft angle $\Delta\theta$. The model also integrates the state equations representing the angular acceleration of the engine (4.62) and turbocharger (4.48) together with the state equations which represent the dynamics of the control actuators (4.58).

When the engine model is computed using a conventional computer system, the state equations for each of the control volumes

have to be computed sequentially and then the rates integrated (also sequentially), as shown in Figure 5.1. A different procedure is adopted when computing the engine model in parallel, in that each control volume evaluates the state equations, and also performs the integrations. This has the obvious advantage that the control volume state equations are evaluated and integrated in parallel, which makes the best use of the processors. Before giving a detailed description of the solution procedure, a brief description will be given of the numerical integrator chosen to integrate the state equations and the general solution procedure, since this choice has a major influence on how the model is computed.

5.3.1 Integrator and General Solution Procedure

A numerical integrator has to be chosen for integrating the engine model state equations. Despite the extensive use of filling and emptying models and their high computational requirements, somewhat surprisingly, the choice of numerical integrator for the control volume state equations does not seem to have been the subject of much review. The work referred to most often in the literature, is that of Annand [5.1], who compared the performance of a predictor-corrector and Runge-Kutta numerical integrator and found the predictor-corrector to be faster for a given degree of accuracy.

A modified Euler predictor-corrector method has been used extensively by the School of Mechanical Engineering [5.2,5.3] and elsewhere [5.4,5.5], to integrate the control volume state equations. Modified Euler was also adopted in this research in

order to make possible an assessment of the improvement in execution speed brought about by computing the engine model in parallel. The choice of integration method is perhaps an area which would warrant further investigation.

Modified Euler is a simple predictor-corrector numerical integrator, employing the following predictor and corrector formulae:

o predictor

$$Y^{p}_{\theta_{n+1}} = Y_{\theta_n} + \Delta\theta \left.\frac{dY}{d\theta}\right|_{\theta_n} \qquad (5.1)$$

o corrector

$$Y^{c}_{\theta_{n+1}} = Y_{\theta_n} + \frac{\Delta\theta}{2}\left[\left.\frac{dY}{d\theta}\right|_{\theta_n} + \left.\frac{dY}{d\theta}\right|_{\theta_{n+1}}\right] \qquad (5.2)$$

$$\Delta\theta = \theta_{n+1} - \theta_n \qquad (5.3)$$

The general procedure for solving the control volume state equations is illustrated in Figure 5.2. The calculations to advance the gas solution by a small step in crankshaft angle (from θ_n to θ_{n+1}) commence with the application of the predictor formula (5.1), to the gas rates for all the control volumes, to obtain estimates of the state variables at the crankshaft position θ_{n+1}. These predicted state variables are then used to evaluate the control volume state equations at θ_{n+1} and the results are substituted into the corrector formula (5.2) to obtain corrected estimates of the state variables at θ_{n+1}. The difference between the predicted and corrected value of each state variable is tested against a specified stability criteria, and if any is outside the acceptable limit (in

any control volume) then all the state equations are evaluated again (using the latest state estimates and gas conditions) and the corrector is applied again to obtain more accurate estimates of the state variables. This procedure is repeated until successive estimates of the state variables converge to the required accuracy, or the corrector calculations have been applied a specified number of times. This process is shown in Figure 5.2. If the corrector formula has been applied for the maximum allowed number of times, and the stability criteria has still not been achieved, then the integration step length is reduced and the process repeated. Having achieved the required degree of accuracy in all the control volumes, the model is advanced by another small step in crankshaft position to perform calculations for that new position.

The state equations which represent the angular acceleration of the turbocharger shaft (4.48) and engine crankshaft (4.62), and the state equations which represent the dynamic behaviour of the control actuators (4.58) also have to be integrated. The physical systems represented by these equations have much longer time constants than the control volume state equations. However, the integration step size used to solve them is the same as that used for the control volume state equations, and is small enough to permit the equations to be integrated without requiring iteration [5.6]. Accordingly, Equations (4.48), (4.58) and (4.62) were integrated using the predictor Equation (5.1).

5.3.2 Parallel Computing Solution Procedure for the Engine Model

The general solution procedure of the engine model can be broken down into five basic stages, namely: increment crankshaft phase, predictor calculation phase, corrector calculation phase, stability phase and finally the update phase. The predictor, corrector and update phases can be computed concurrently and constitute most of the computational effort of the solution procedure. The method is discussed in detail below, commencing at the increment crankshaft phase (stage 1) and assuming that all model conditions (including rates of change) are known at a crankshaft position θ_n.

It is required to advance the model solution by a small step in crankshaft position to a new value θ_{n+1}. A flow diagram illustrating the responsibilities of the supervisor task in the solution procedure is shown in Figure 5.3. Details of the calculations performed by the control volume tasks, actuator task and engine shaft task are given in Chapter 7.

- stage 1, increment crankshaft angle phase: Calculations commence to advance the engine model solution by a small step in crankshaft position when the supervisor task sets the "next" crankshaft position, at which an attempt is to be made to solve the model equations (ie θ_{n+1}) to the current crankshaft position (θ_n) plus the integration step size ($\Delta\theta$). ie $\theta_{n+1} = \theta_n + \Delta\theta$

- stage 2, Predictor calculation phase: The supervisor task sends each control volume task a command to integrate the gas

rates ($dT/d\theta$, $df/d\theta$ and $dm/d\theta$) at crankshaft position θ_n, (using the predictor formula (5.1)), to obtain estimates of the state variables at θ_{n+1}. The integration step size to be used in the calculations is specified by the supervisor task when it sends the "predictor" command. As each control volume task completes its predictor calculations, it informs the supervisor task that it has done so, and returns the predicted state variables and other gas conditions at crankshaft position θ_{n+1}: these are required by connected control volumes as boundary conditions, during stage 3 of the calculation process.

At the same time that the supervisor task sends each control volume task a command to perform the "predictor" calculations, it sends a command to the control actuator task and engine shaft task. This command is to evaluate and integrate their state equations, in order to obtain values at the new crankshaft position θ_{n+1}, for the engine model control inputs (static timing, fuel shot mass and turbine nozzle restriction), engine and turbocharger speed. Having completed their calculations, the shaft task and actuator task return the results to the supervisor task for use during the stage 3 calculations. Once the supervisor task has received the results of the predictor calculations from all the simulator tasks, stage 3 calculations can commence.

- stage 3, Corrector calculation phase: The supervisor task sends each control volume task a command (and the appropriate boundary conditions), first to evaluate their state equations

at crankshaft position θ_{n+1} and then apply the corrector integrator formula (5.2) to obtain improved estimates of the state variables (at θ_{n+1}). Having done this, each control volume task compares the difference between the latest and previous estimates of its state variables (at θ_{n+1}) against the allowed tolerance, and if any state is outside its limits, a stability flag is set to indicate a failed result. As each control volume task completes the "corrector" command calculations, it informs the supervisor task and returns the latest state estimates, other gas conditions (required if the corrector formula has to be applied again - see stage 4) and the stability flag. Once all the control volume tasks have completed the stage 3 calculations, the solution procedure advances to stage 4.

- stage 4, stability phase: As Figure 5.3 shows, the next command sent by the supervisor task depends upon the stability results just returned by the control volume tasks. If any control volume task returned a failed result and the corrector calculations (stage 3) have not been applied the maximum allowed number of times (three was used in this research [5.7]), then the solution procedure continues with another application of the corrector calculations at stage 3. However, if a failed stability result was returned and the stage 3 corrector calculations have been applied the maximum allowed number of times, then the supervisor task reduces the integration step size in use (by half in this research) and the solution procedure restarts at stage 1. When all the

control volume tasks return a "pass" stability result, the solution progresses to stage 5.

- o stage 5, update phase: The supervisor task sends a command to control volume tasks, control actuator task and engine shaft task to update their state variable values, which relate to the previous crankshaft position (θ_n), to the values for the new crankshaft position (θ_{n+1}). Finally, the supervisor task loops back to stage 1 to advance the engine model solution by another small step in crankshaft position.

5.4 Task Scheduling

The previous sections of this chapter have described how the filling and emptying engine model can be divided along thermodynamic control volume boundaries and computed in a parallel fashion. Having identified concurrency in a problem, an important aspect which then has to be decided is how the computational tasks should be scheduled among the processors, to compute the problem as quickly as possible. If, as is usual, all the processing nodes are identical, then one way of achieving the fastest execution time is to allocate one processor to each computational task. Whilst this results in the fastest computation, some of the computational tasks may be trivial by comparison to others and consequently the computational efficiency overall may therefore be poor. Depending upon the nature of the calculations, it may be possible to schedule the computational tasks in such a manner that some processing nodes compute more than one task without any degradation occurring in the

overall computational time. This would be possible, for example, if several of the more trivial tasks could be computed serially by one processing node, in a time which is no longer than the time taken by the node which is responsible for computing the longest computational task. Clearly, such a solution is much more elegant and has a higher computational efficiency.

Careful consideration should always be given to obtaining the best method for scheduling the computational tasks. Cost consideration must be borne in mind, and a compromise may have to be made between the conflicting requirements of attaining the fastest computational speed, and the number of processors available. If a compromise has to be made, an examination of alternative methods of scheduling the computational tasks is inevitable. Unfortunately, determination of the scheduling scheme which will result in the fastest computational speed in these cases, is not generally amenable to accurate evaluation and often much reliance must be placed on "good engineering intuition".

Two common methods of scheduling are dynamic task allocation and static or pre-determined task allocation. With dynamic task allocation, work is allocated to the processing nodes whilst the problem is being computed. For example, in its simplest form, the dynamic allocation would be to use a single processor to supervise the work load on the computer system and to dispatch tasks to the processors during run time, in such a manner as to ensure the fastest computational speed. The basic problem which the dynamic scheduler has to solve is, given a number of tasks to compute, each having a different execution time, how best should it schedule the

tasks on to the available processing nodes to compute the application in the minimum possible time ? One way in which the dynamic resource allocation problem can be solved is to use the project management technique PERT. With this technique, the dynamic scheduler evaluates the times in which the tasks can be computed using all possible options for the task allocations, and then implements the scheme with the minimum execution time (critical path).

The static or pre-determined approach to task allocation specifies prior to carrying out the computations, which tasks each processor is to compute, and has the considerable advantage of not having the computational overhead inherent in the dynamic scheduler. Because of this, a static task schedule can result in a faster computational speed than could be achieved using a dynamic scheduler (although this depends very much on the application). The static scheduler also has the considerable practical advantage that it is a much easier scheme to implement than even the simplest dynamic scheduler, although this consideration should be of secondary importance.

5.4.1 Task Scheduling for the TL11 Engine Model

Nine control volume tasks (six cylinders and three manifolds) a control actuator task and an engine shaft task are required to represent the dynamic behaviour of the TL11 engine. In addition, a supervisor task is required to manage the engine model solution procedure; thus in total, the TL11 engine model uses 12 tasks.

Unfortunately, financial considerations restricted the capability of the parallel computer system developed for the research (see next chapter) to the use of five slave processing nodes and an input/output processor (‡). Clearly the number of computational tasks (12) exceeds the number of processors available (5) by a considerable margin, and careful consideration had to be given to the best method of computing the tasks, so that the fastest execution speed could be achieved.

If a reasonable computational balance between the processors can be achieved using a static task allocation, then it should be used. The major problem when calculating the engine model using a static allocation is that the calculations for computing a cylinder control volume task vary very considerably between the phases of the engine power cycle (e.g scavenge, induction, compression etc). In particular, the rate at which fuel burns has to be calculated only during the combustion phase of the engine power cycle, and the valve flows only have to be calculated during the open phases (scavenge, induction and exhaust). Despite this, by using prior knowledge of the engine model calculations, it is possible to devise a static allocation scheme which should produce a reasonable computational balance. An inspection of the calculations required for the different phases of a cylinder control volume (see Chapter 4) shows that the open phases of the engine cycle require more computation than do the closed phases. This is because during the open period, the flow of gas through each valve has to be evaluated and then

(‡) For the reasons given in Chapter 6 the input/output processor does not participate in the engine model calculations.

included in the calculation of the gas state equations. The cylinders of the TL11 engine are arranged in pairs, located at the same crankshaft position, but offset from one another in the engine power cycle by a complete revolution of the engine. Thus, when one cylinder is operating in the open period of the engine cycle, the other is operating in the closed period, and vice versa, - (except for the periods of overlap which occur at the transitions between the open and closed phases). This characteristic of engine behaviour led to the tentative conclusion that using a processing node to compute both cylinders should result in a reasonable computational balance, since each processor is then responsible for computing at least one, but at most, two valve flows. It was decided that this static allocation scheme should be tried and evaluated before considering whether it would be worthwhile to develop a dynamic scheduler bearing in mind the considerable effort which would be involved (‡).

The static task allocation used to compute the TL11 engine model is shown in Figure 5.4. As the figure shows, three slave processing nodes are used to compute two cylinder control volume models, one processor is used to compute the two exhaust manifold control volumes and the last slave processor is used to compute the remaining model tasks. These remaining tasks are the supervisor task, shaft task, control actuator task and the inlet manifold task.

(‡) In fact, as is shown in Chapter 9, an adequate computational balance was achieved using this static task allocation.

5.5. Look Up Tables

Although major improvements in execution speed are expected by computing the engine model in parallel, the use of look up tables was also investigated as a means of making the engine model run even faster. Unless a computer system incorporates hardware dedicated to performing floating point operations, these operations have to be performed by software - and are very demanding computational requirements. In particular, the evaluation of functions such as sinusoids, cosinusoids, powers, logs etc all require extensive computation and if look up tables can be used as an alternative, then the speed of computation should be much improved. Even greater improvements may be possible if solutions for complete, or parts of equations can be stored in look up tables. For example, the volume of gas trapped in a cylinder (4.49) can be stored in a look up table as a function of crankshaft angle. Then, when the volume of gas is required, it can be obtained very quickly by making the appropriate entry into the table and with a substantial saving in computation, - even if the computer system has dedicated hardware to perform floating point operations.

The disadvantage of look up tables, is that limited memory does not permit storing the function or equation with as good a resolution as it can be calculated and analysis of the error incurred in the overall accuracy of the engine model is difficult to assess.

In view of the significant improvement in execution speed which was expected to be realized, it was decided to develop a

version of the engine model using look up tables.

5.6 Simulator Requirements

Although the primary objective of this programme of research is to evaluate the feasibility of solving the diesel engine filling and emptying engine model in parallel, it was also considered highly desirable to produce an engine model which is of practical value. This required the development of a diesel engine simulator, based of course, on solving the filling and emptying engine model in parallel. The general requirements for the simulator are described below.

To make the simulator as easy as possible to use and to increase realism, it was decided to make operation of the simulator as near as conveniently possible to the operation of a real engine. The best way of doing this is to design the simulator software so that once it is running on the computer system, the user can interactively change the engine controls and/or engine load at any time, to drive the engine model to a desired operating condition, - as though operating a real engine, albeit in slower than real time.

When driving a real engine, the operator is often presented with details of current engine performance, such as engine speed. Accordingly, to further enhance realism, it was decided that the simulator should use a graphics system which would display continuous updates of engine model performance. The operator is then able to see immediately the effect of changes in the engine controls on engine model behaviour.

Most applications will require the simulator to record engine model performance results, and display the results on the graphics system and/or store the results on disc for subsequent analysis. Because the simulator will allow the operator to interactively drive the engine to different operating conditions and change engine control settings as desired, it was decided that the data logging system should also provide this degree of flexibility. Thus the simulator should allow the operator to switch data logging on or off, display and/or save engine responses, as and when desired. This flexibility allows the operator to drive the engine to a desired operating condition, then subject the engine model to a test signal, and record, display and/or save the engine responses, and then drive the engine to a different operating condition and carry out further tests without having to stop the simulation between test runs, - once again as though performing experiments on a real engine.

The simulator should have sufficient memory capacity to make possible extensive data logging of engine responses, including the recording of fast changing quantities such as the gas conditions in the control volumes over a number of engine cycles, and also the recording of quantities such as engine and turbocharger speed (which vary much more slowly), over many seconds of real time. In addition to recording engine time responses, there are many applications in which the simulator has to evaluate and display steady state engine model performance.

5.7 Summary

The division of the filling and emptying engine model along control volume boundaries, into a number of tasks which can be computed in parallel has been described. This division was chosen because the control volume models are the principal computational tasks of the filling and emptying model; also they are relatively independent, and the division results in a neat and flexible program structure. The modified Euler numerical integrator has been chosen for integrating the control volume state equations, to allow a comparison with the performance of other filling and emptying engine models running on conventional computer hardware. The solution procedure is managed by a supervisor task which instructs the engine model tasks when they should perform the predictor and corrector calculations and when they should update their variables.

The engine model is computed using a static task allocation, based on a prior knowledge of the calculations required for the engine model tasks, so as to result in (as best as possible) an equal computational load between the processors.

Finally the general capabilities of the engine simulator have been described. These include a flexible user interface to allow the operator to "drive" the simulator, record and display engine responses as required, and an animated display of engine model performance.

5.8 References

5.1 W J D Annand.
Choice of a Computing Procedure for Digital Computer Synthesis of Reciprocating Engine Cycles.
1968, Journal of Mechanical Engineering Science, Vol 10 no 3.

5.2 S Charlton.
Spice, Simulation Program for IC Engines (User manual).
April 1986, School of Mechanical Engineering University of Bath.

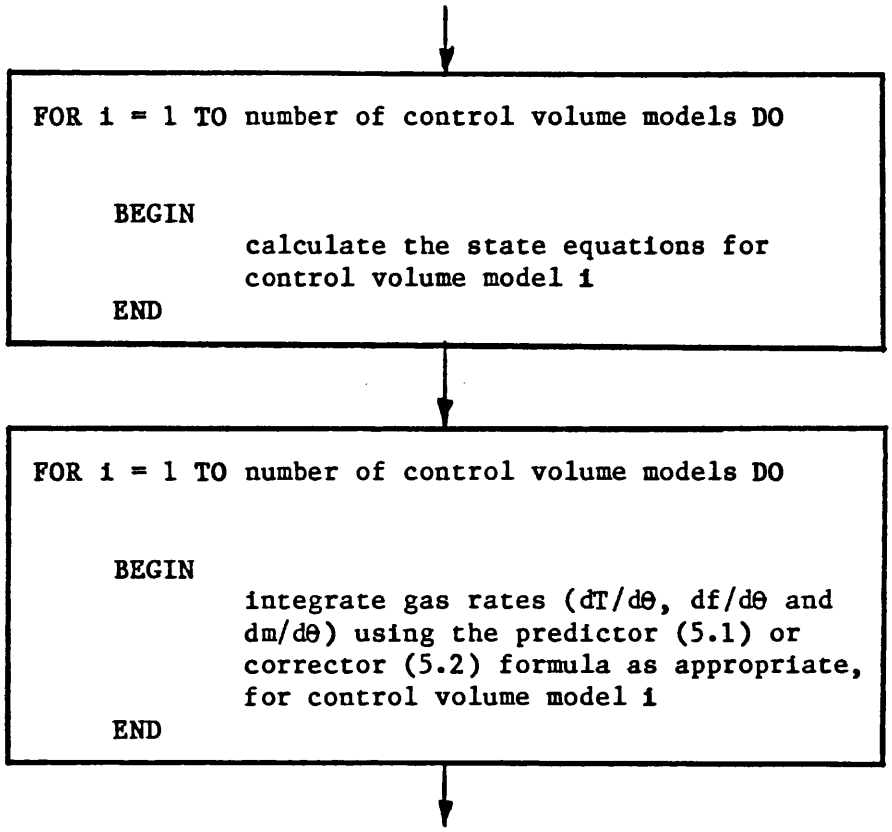
5.3 M Tarabad.
Diesel Engine Cycle Simulation.
Sept 1983, School of Mechanical Engineering University of Bath.

5.4 K J McAulay, W Tang, S K Chen, G L Borman, P S Myers, O E Uyehara.
Development and Evaluation of the Simulation of the Compression Ignition Engine.
1965, SAE 650451.

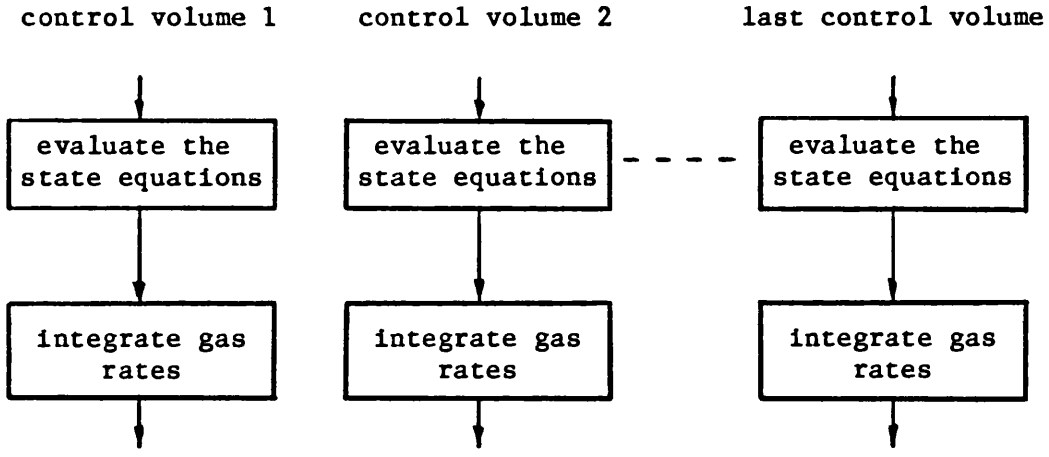
5.5 E E Streit, G L Borman.
Mathematical Simulation of a Large Turbocharged Two-Stroke Diesel Engine.
1971, SAE 710176.

5.6 M Marzouk.
Simulation of Turbocharged Diesel Engine Under Transient Conditions.
1976, PhD Thesis University of London (Imperial College).

5.7 C F Gerald.
Applied Numerical Analysis Second Edition.
1978, Addison-Wesley Publishing Company.



o Sequential Solution



o Parallel Solution

Figure 5.1 Sequential and Parallel Calculation and Integration of Control Volume Models.

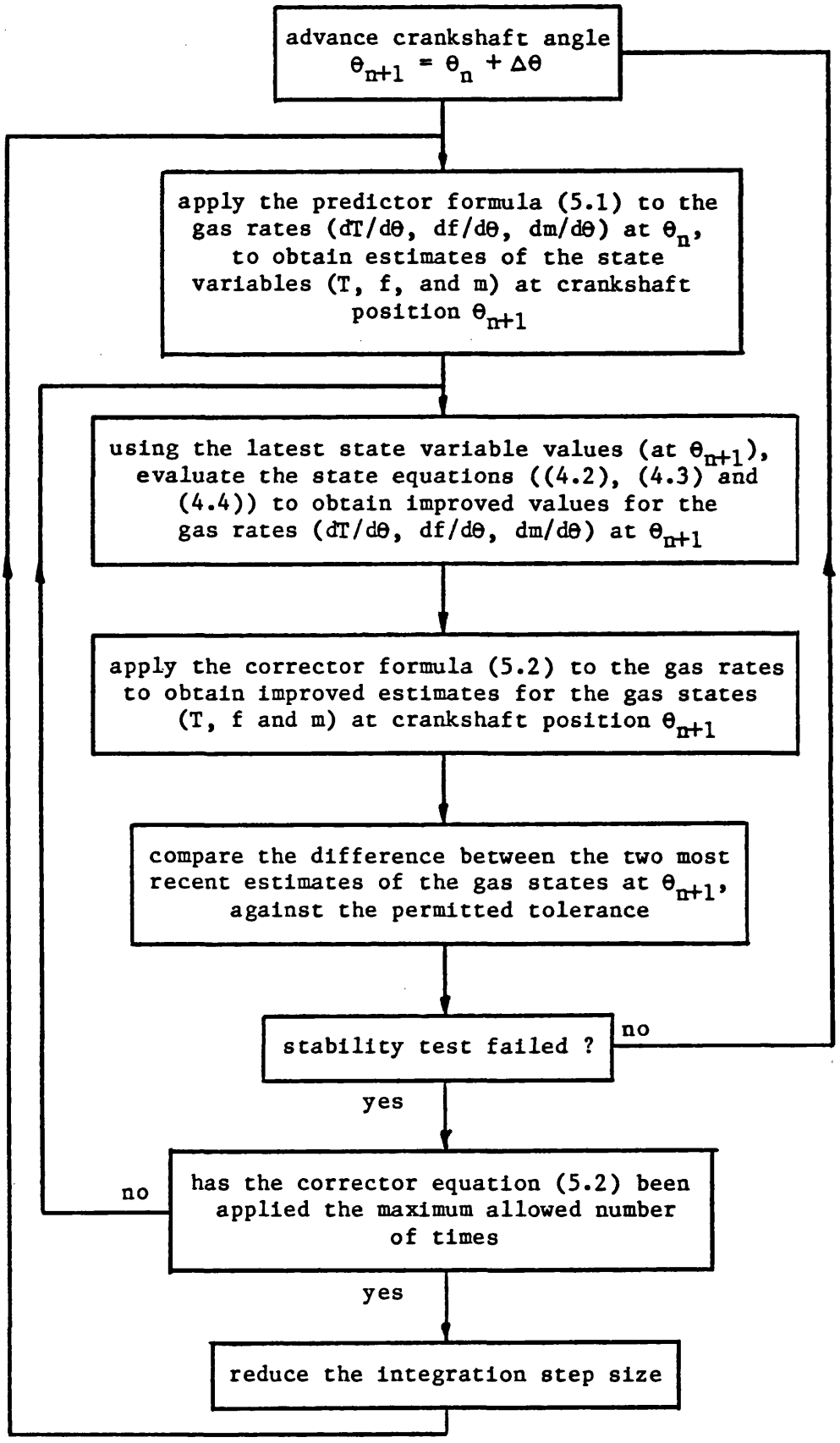


Figure 5.2 General procedure for solving control volume state equations

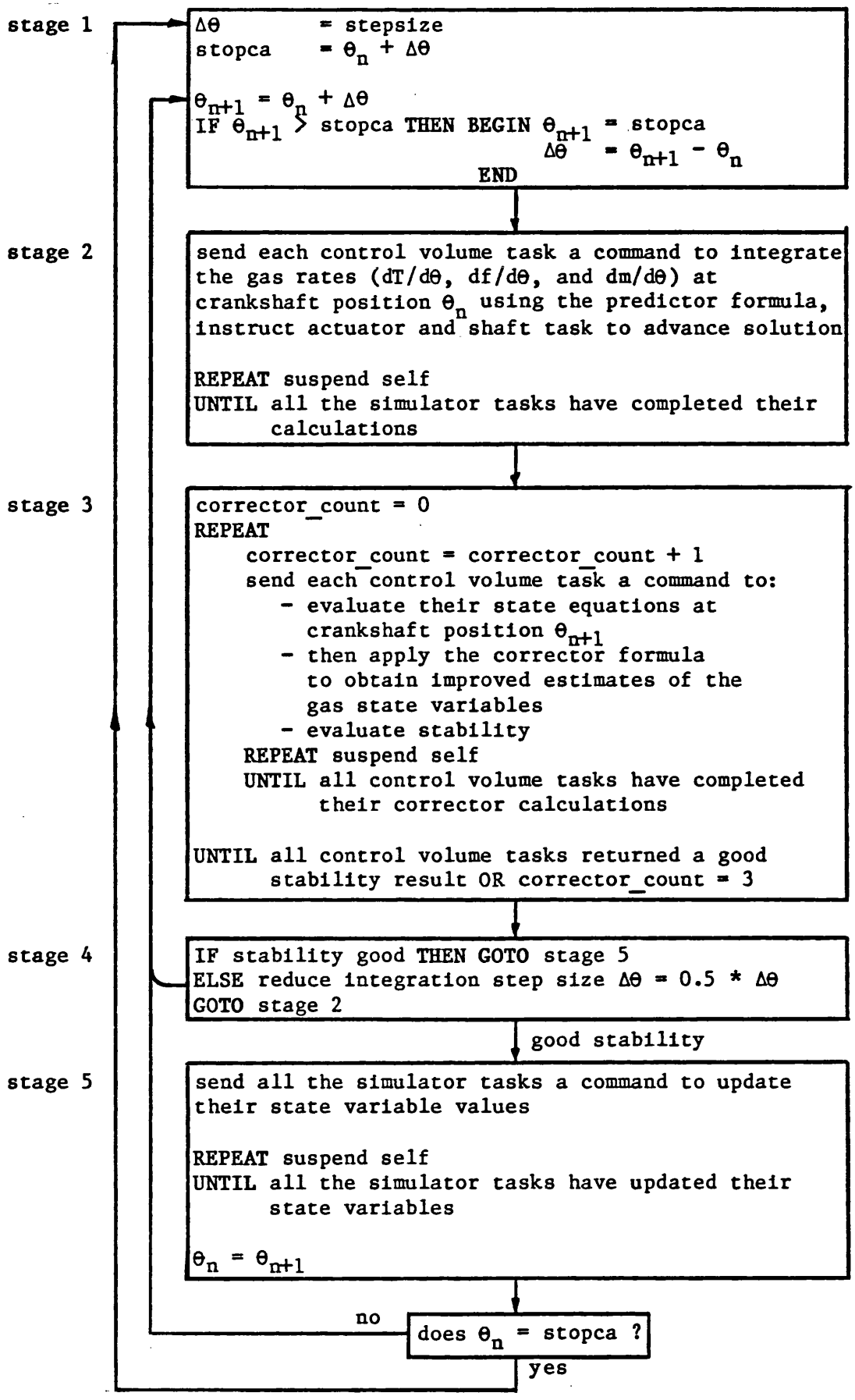


Figure 5.3 General Operation of the Supervisor task.

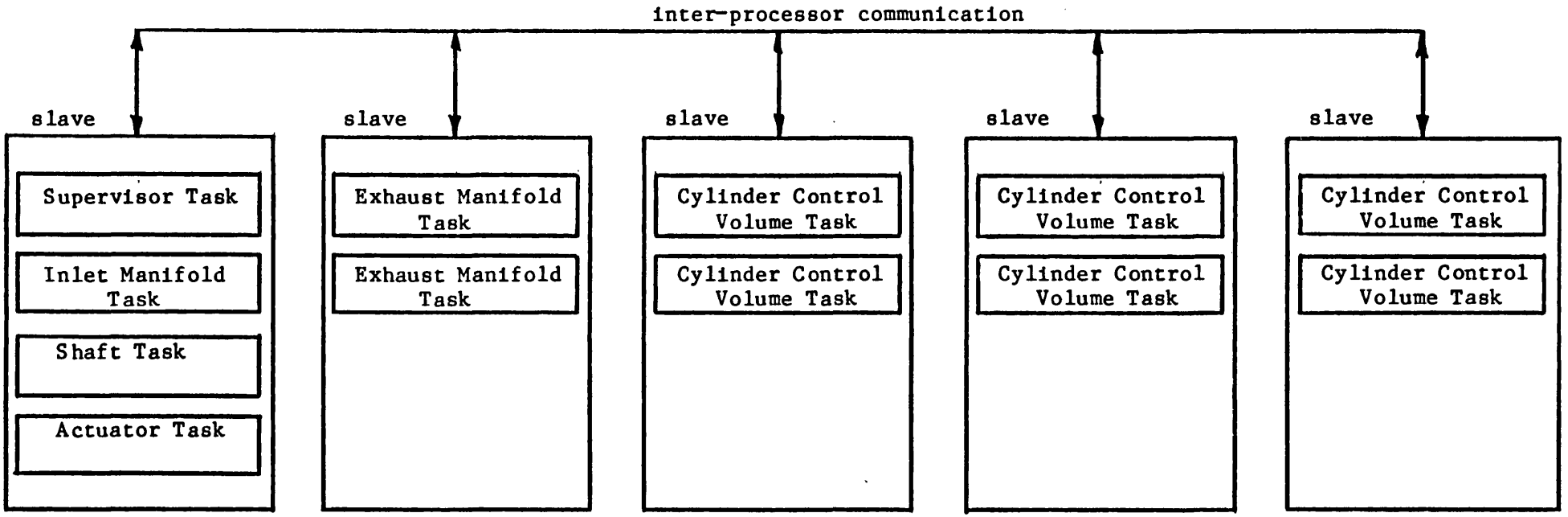


Figure 5.4 Task Allocation used to compute the TL11 Engine Model.

CHAPTER 6

6.1 The Computer System

This chapter describes the parallel computer system hardware and operating system software used to compute the filling and emptying engine model. The most suitable computer system architecture for use with the engine model is described in Section 6.2, the computer system hardware is described in Section 6.3 and the operating system software is described in Section 6.4. A much more detailed description of the computer system hardware and operating system software is given in Reference 6.1.

6.2 Computer System Architecture

Many parallel processing architectures have been proposed ranging from the linking together of several computers which individually are Von-Neumann machines (‡), to much more radical approaches such as data flow machines (which are sometimes referred to as "non-Vons" - i.e not Von-Neumann architectures). Most parallel computer architectures can be characterised as follows:

(‡) Almost all computers are built using the so called Von-Neumann architecture. They perform one operation at a time using a single processing element on a single stream of data, and are known as single instruction, single data stream, machines (SISD).

- **Single Instruction Multiple Data stream (SIMD) architecture.** In these machines a single instruction causes a number of data streams to be acted on in parallel. The best known machines of this kind are array processors which are designed to speed up the calculation of vectors, matrices and arrays of numbers, all of which have highly repetitive arithmetic operations. Array processors are often attached to conventional SISD stream machines to improve the performance of the machines when carrying out these tasks.

- **Multiple Instruction Multiple Data stream (MIMD) architecture.** These machines permit different data streams to be computed concurrently using different instructions, and are therefore generally more useful than SIMD stream machines.

The most suitable architecture for computing the filling and emptying engine model in parallel, is a MIMD system. This is because the filling and emptying model is very amenable to being broken down into a number of separate and relatively independent control volume tasks, each task solving a different set of equations (eg induction, compression, manifolds etc), and which can be computed by separate processing nodes, as was described in the previous chapter.

Multi-processor (MIMD) computers are broadly defined as loosely or tightly coupled, and differ fundamentally in the manner in which they provide inter-processor communication. If each processor is relatively autonomous and communicates using shared communication channels the system is said to be loosely coupled.

Such processors can access their own local memory directly, but can only access another processor's memory indirectly, by sending a message to the receiving processor, requesting it to perform the access on their behalf. In a tightly coupled system, inter-processor communication is characterised by a common memory which can be accessed by any processor using a shared data bus. In such a system, any processor can directly access or change information quite simply by executing a read or write instruction to the area of shared memory.

The advantages of the tightly coupled system are flexible and fast inter-processor communication, and a good program debug capability, which can be very suitable for debugging complex software (such as the engine model) which runs in unison on several processors. Although a powerful debug facility makes software development much easier, it does not necessarily make the computer system more suitable for computing the engine model, and the debug capability should be seen as a secondary consideration when deciding which type of computer system to use.

The most important advantage of the tightly coupled system is the speed at which it can exchange information between processors. This can be by reference, which means that processors physically exchange a pointer showing where the data is stored in memory; in this way complex data structures (for example arrays of data logged engine model responses) can be exchanged quickly. Transfer of complex data structures using the loosely coupled system is much more complex, and generally much slower, since the data has to be copied from one processor to the other.

The main disadvantage of the tightly coupled system, is that it is less tolerant to processor failure. If a processor fails in a tightly coupled system, the complete system may be rendered unusable, whereas a loosely coupled system can be designed to detect a failure and continue to operate - albeit at a reduced speed by avoiding the use of the failed processor. This advantage of a loosely coupled system was considered to be of little importance in this research and in view of the advantages of a tightly coupled system, this type was chosen for computing the diesel engine model.

6.3 Computer Hardware

6.3.1 General Processor Requirements

The division of the engine model along control volume boundaries and the general capabilities to be provided by the engine simulator (Chapter 5), define the general requirements of the computer system hardware. Nine control volume tasks are required to represent the behaviour of the TL11 engine and the calculation of these nine tasks constitutes the vast majority of the computational effort involved in computing the engine model. It would have been ideal if nine processing nodes could have been available so that each processor could have been made responsible for computing one control volume task. Unfortunately, because of financial considerations this was not possible, and as has been explained in the previous chapter, a compromise had to be sought between achieving the fastest execution speed and cost. In arriving at this compromise the next logical step in allocating the engine model

tasks was to use each processing node to compute no more than two control volume models each; this required five processing nodes (in fact, for reasons given below, a sixth processor was required) and was possible within the available funding.

In addition to computing the engine model equations, the computer system has to provide additional facilities such as graph plotting and also fulfil more basic requirements such as handling computer system input and output to the console, disc drives etc. These additional facilities require the use of a processor, and if they are carried out by a processor which is also responsible for performing engine model calculations, the execution speed of the engine model will be significantly reduced. To avoid this it was decided to use a sixth processor to provide the computer system IO and the simulator facilities which are not directly related to computation of the engine model equations. This processor is referred to as the IO processor. Thus, in total, the computer system consists of an IO processor and five slave processing nodes.

6.3.2 Description of the Hardware

The computer system chosen for computing the engine model was developed in the Department [6.1] and uses single board processing nodes based on the Motorola MC68000 microprocessor [6.2]. A block diagram of the computer system is shown in Figure 6.1; it consists of six processing nodes, a memory card, a graphics card, a backplane display card, a backplane arbiter card and a local area network card. A photograph of the system is shown in Figure 6.2, showing

the main console which houses the electronics, power supplies and disc drives, the graphics display monitor and terminal. Each element in the computer system will now be discussed.

6.3.2.1 Backplane System Bus, Backplane Display Card and Backplane Arbiter Card

Physical connections between the various cards in the computer system are made via the backplane into which they are plugged: the backplane also carries the system bus. The system bus is a critical element in the computer system since all communication between processors and other resources take place using it. The system bus was specifically designed to be compatible with the Motorola MC68000 family of microprocessors. It is compatible with the previous system bus used by other computers developed in the Department [6.3, 6.4], which gives it the considerable practical advantage that it can be used with a number of existing circuit boards, such as the graphics card and memory card. The system bus signals are listed in Reference 6.1.

The system bus front panel display can be seen at the top of the main console on the computer system photograph in Figure 6.2. It consists of a number of light emitting diodes (LED's), one for each data and address line, and one for most of the system bus control signals. The backplane display card monitors the state of the various system bus signals and switches on the appropriate LED when any activity takes place. Thus the intensities of the various LED's, give a crude measure of the system bus usage.

Because the processing nodes all share the same system bus, any node wishing to use the bus must first request permission to do so from the backplane arbiter card. If the system bus is not being used, the arbiter grants the processor permission immediately; if the bus is in use, the requesting processor has to wait until the bus becomes free, when the arbiter card then gives permission. If more than one processor asks (or is waiting) to use the bus at the same time, the arbiter decides which processor shall be given use of the bus according to predetermined priorities.

6.3.2.2 The MC68000 Processing Node

The computer system employs six processing nodes. One processor, designated to be the input output (IO) processor, is responsible for all the systems input and output requirements i.e to the hard disc drive, floppy disc drives and the console. This processor also performs all the supervisory functions which are required for multi-processor computing, such as loading code into the other processing nodes and instructing them when to commence execution: it is also used for the general software development. The remaining five processors are used solely to perform engine model calculations. A photograph of a slave processing node is shown in Figure 6.3.

A block diagram of the processing board is shown in Figure 6.4 which shows that the board consists essentially of a processor unit, memory, processor control register, IO and a system bus interface. Each of these elements will now be discussed.

- Processor: Each processing node employs a fast version of the Motorola MC68000 microprocessor [6.2] having a clock speed of 12.5 MHz. The MC68000 uses 16, 32 bit wide general purpose registers for internal operations and can perform operations on bits, bytes (8 bits), words (16 bits) and long words (32 bits). Externally, the MC68000 uses a 16 bit data bus and a 24 bit address bus which can directly address 16 Mbytes of memory.

  The MC68000 microprocessor provides a test and set (TAS) instruction which uses a "read modify write memory cycle" and this considerably simplifies the use of the MC68000 in a multi-processor system. Essentially, during a read modify write memory cycle, the processor retains control of the shared system bus for the whole of the time the instruction is being executed, so that it can read and modify a flag in one and the same operation. These flags (referred to as semaphores) are used to indicate whether a resource is in use. Consequently, a processor can read the semaphore and (if the resource is not in use), modify the semaphore during the same instruction cycle. This enables a processor to reserve a resource for its own subsequent use without there being any danger that another processor is already in the process of reserving the same resource.

- Memory: Each processor has sufficient on-board memory to store its local operating system code, application program code and data. This has been done to avoid a processor having to make considerable use of off-board memory (which would

increase traffic on the system bus, and generally result in processors having to wait longer before gaining permission to use the system bus). Because of the many tasks which the IO processor has to perform, it has the greatest requirement for memory and accordingly was equipped with one Mbyte of on-board dynamic random access memory; the slave processing boards were equipped with 256 Kbytes of memory. The on-board memory was designed to allow local memory cycles to take place without having to insert wait states, so that the processor could operate at maximum speed.

The memory used by each processor occupies a unique area in the MC68000 address space and was designed to be dual ported so that the processors can exchange information. The dual ported memory basically works as follows: an off-board processor, which has already received permission to use the system bus (from the bus arbiter card), signals its intention to perform an on-board memory access by issuing a local bus request signal. The off-board processor must then wait for the receiving processor to grant it permission to commence the local bus access, during which time the off-board processor retains control of the shared system bus. In order to minimise the time that the system bus is unavailable to other processors, the receiving processor treats the incoming request with the highest priority, and once it has completed its current cycle grants the off-board processor permission to perform its access by issuing a bus grant signal. The off-board processor then proceeds to perform the local memory

access. On completion, the accessed processor regains control of its local bus and the shared bus is made available for other processors to use.

In addition to its random access memory (RAM) the IO processor has some electrically programable read only memory (EPROM) which is used to hold a bootstrap program.

- **Processor control register:** The processor control register enables a processor to monitor and control the operation of another processor. For example, the debug system (see Section 6.4.4) uses a processor's control register to monitor the execution of a program running on the processor. The control register also provides the means whereby a processor can gain the attention of another processor (by interrupting it) when seeking to exchange information.

  The control register has four bits; these perform the following functions, halt (or continue) the processor, interrupt the processor, reset the processor and enable (or disable) the ROM reset vector. The state of a processor can be determined by reading its control register.

- **Input Output (IO):** In order to limit the number of accesses occurring using the shared system bus, each processing node was given its own on-board IO capability, rather than using separate IO boards plugged into the backplane. This had the added advantage that it maximised the number of backplane slots available for processing nodes.

Provision was made for the IO facilities listed below, although only the IO processor used them and therefore was the only board populated with the necessary integrated circuits. The IO facilities were: two RS232 serial communication channels, floppy disc controller, hard disc controller and a direct memory access (DMA) controller. All processing nodes were equipped with a timer which, on the IO processor, was configured as a real time clock.

- o System bus interface: The bus interface is responsible for coupling a processor local bus to the shared system bus. The majority of the circuitry consists of bi-directional tri-state buffers which enable inwards or outwards memory cycles depending upon the type of access being executed. The principle accesses which the bus interface has to supervise are, a local processor performing a memory cycle off-board, and an external processor performing a local memory access.

6.3.2.3 Memory Card [6.3]

This provides 256 Kbytes of memory which can be accessed via the system bus, and is used as a general purpose storage area.

6.3.2.4 Graphics Display Card [6.4]

The graphics display card provides a high resolution colour display and is accessed via the system bus. It uses the EFCIS (EF3965) colour graphics controller which controls two pages

of graphics information. Each page has a resolution of 512 by 512 pixels and can use eight colours. One graphics page is displayed while the other is being updated; a smooth moving display is produced by continuously updating the non-displayed page and then switching pages.

6.3.2.5 Local Area Network Card [6.1]

This supports the "multilink" local area network which provides a common data link between network stations connected together in a ring. The computer system uses this network to access printers, a pen plotter and other computers connected in the ring.

6.3.2.6 MC68020 Based Processing Node

The previous sections, have described the computer system hardware which was used for the majority of the engine modelling activity, and which uses processing nodes based on the MC68000 microprocessor. Since releasing the 16-bit MC68000 microprocessor in 1979, Motorola have designed a significantly more powerful member of the MC68000 family of microprocessors, namely the 32-bit MC68020 [6.5] microprocessor, which they released in 1984. Use of this processor as a means of further increasing the speed of computing the engine model was obviously of great interest and three processing nodes each incorporating a MC68020 chip have been built for evaluation with the tightly coupled computer system.

The MC68020 processing node hardware was designed to be compatible with the existing computer system, since this worked well and compatibility with it would permit the continuing use of the computer hardware already designed and built. To maintain upwards compatibility with the MC68000 based hardware, the MC68020 processing node had to support the same system bus interface, the same shared memory mechanism and the same processor control functions. The computer system could then operate quite happily using a mixture of MC68000 and MC68020 based processing nodes. Indeed this was essential, since the MC68020 processor board was designed specifically to operate as a fast "number cruncher" and was only provided with the minimum IO required to run a monitor program for hardware debugging: all other computer IO requirements continued to be met using a MC68000 based IO processor.

A block diagram of the modified computer system is shown in Figure 6.5. The bus arbiter card, MC68000 based IO processor, local area network card, graphics card, memory card and system bus display card have already been described. The three MC68020 processor boards were built by wire wrapping and a photograph of one of them is shown in Figure 6.6. The processing node is the same size as the MC68000 processing node, although its chip count is lower. A block diagram of the MC68020 processing node is shown in Figure 6.7; this shows that the processing board consists of a MC68020 processor, a MC68881 co-processor (which performs floating point calculations), local memory, a processor control register, IO and a bus interface. Each of these will now be briefly discussed.

- MC68020 processor [6.5]: Internally, the MC68020 uses 16, 32-bit general purpose registers, a 256 byte high speed instruction cache and is object code compatible with other members of the MC68000 family of microprocessors; it also includes some new and enhanced instructions. The instruction cache increases processor performance, since the processor can fetch instructions from cache more rapidly than it can from main memory. The resulting improvement depends on which code is being executed, and varies from no improvement for in line code, or code with long loops (in which case all instructions still have to be fetched from main memory), to almost doubling the speed when executing code which is in close locality to reference, e.g short loops (which can be run entirely from the cache store). Since the cache only stores instructions, data accesses must still be made from main memory.

  The MC68020 uses a 32-bit data bus and a separate 32-bit address bus which can directly address 4 Gbytes of memory. It uses dynamic bus sizing which makes it easier for the designer to interface it to 8, 16 or 32 bit devices, and also supports a general co-processor interface. The processor board was designed to operate using the fastest (25MHz) MC68020 microprocessor devices. However, when the boards were built, the only devices available in this country were "slow" 12.5 MHz (ǂ) versions and these had to be used.

(ǂ) Since carrying out the experiments the 12.5MHz devices have been replaced by 16.67 MHz devices and it is intended to increase the speed further by using 25MHz devices when they become available.

- MC68881 co-processor [6.6]. Although the MC68020 is a very powerful processor, it does not have all the special facilities that are required for certain applications, and for this reason a general co-processor interface was incorporated in its design. This was used to give the processing node a fast floating point arithmetic capability, by using a single MC68881 floating point co-processor.

  The MC68881 co-processor is a full implementation of the IEEE standard [6.7] definition of data formats for 32-bit (single precision), 64-bit (double precision) binary floating point arithmetic, and an extended precision format (64 bit mantissa, sign bit and a 15 bit signed exponent). The extended precision format is used to store intermediate values when carrying out calculations, before converting the result to the final data format. A list of the mathematical operations which the MC68881 co-processor supports is given in Table 6.1.

  The co-processor interface operates using the so called "F-line" operation code, so named because all the bits in the upper nibble of the instruction are set (i.e FXXXH). When the MC68020 encounters an F-line instruction it calls the co-processor to execute the instruction and while the co-processor is doing this, it starts its next instruction, which further enhances performance.

- Memory: Generally speaking the aim in much simulation work is to compute in real time (or possibly faster). In some applications, the MC68000 processor will not be fast enough to

compute in real time but the MC68020 processor will be - and if that is so, the choice of the MC68020 is clear. However, it may still be advantageous to use the MC68020 in some applications for which the MC68000 is fast enough. For example, if the MC68020 can be given more calculations to carry out, and still perform in real time, it may well be possible to reduce the number of processors required (in a parallel computer system). Use of the MC68020 in this way increases the memory required to store the code and data, and for this reason it was provided with 1 Mbyte of local memory, compared with 256 Kbytes for the MC68000 processors (except for the IO processor which was also provided with 1 Mbyte of memory).

The MC68020 can access memory extremely quickly, requiring only 3 clock cycles compared with 4 for other members of the MC68000 family. When accessing dynamic memory (which has a relatively slow access time) the speed of the MC68020, generally has to be deliberately reduced, in order to give the memory enough time to respond. This is not necessary with static memory (which has a much faster access time), but since the storage density of static memory devices is not as good, the available board space would restrict the maximum local (static) memory for the processing node to 0.5 Mbyte. For this reason, and also because dynamic memory uses less power and is cheaper, it was decided to design the MC68020 board to operate using dynamic memory accessed with one wait state. It should be noted that inserting the wait state only slows down

the speed at which the processor can access main memory and has no effect on the speed of its internal operations, or on the speed at which it can fetch instructions from cache.

The MC68020 processor node, is also equipped with some EPROM so that it can run a monitor program during hardware debugging.

- **Processor control register:** The processor control register was designed to be upwards compatible with the processor control register used with the MC68000 processor node. Consequently, it supports all the MC68000 control register functions, i.e halt (or continue) the processor, interrupt the processor, reset the processor and enable (or disable) the ROM reset vector, as well as an additional function which can enable (or disable) the processor cache.

- **IO:** The MC68020 processing node was equipped with the minimum IO necessary to run a monitor program for hardware debugging. It provides two RS232 serial interfaces and a programmable timer.

- **System bus interface:** The system bus interface, had to be compatible with that used by the MC68000 boards. It supports the same protocols for requesting use of the shared system bus and for performing memory accesses on offboard processors.

6.4 Multi-Processor Operating System

The previous section has described the hardware for two powerful multi-processor computer systems and, as with any computer, a suitable operating system is required to oversee their operation. Clearly the operating system for a multi-processor computer system has to cater for some requirements which are irrelevant when using conventional computers, such as managing the exchange of information between tasks which may be loaded on different processors.

Essentially the user of a multi-processor computer system requires the operating system to perform the following operations:

- IO management; for example, control of disc storage and console handling.
- Task management; for example, supervision of loading, initialization, execution sequence and deletion of a task on any processor.
- Handling communication between tasks which may be loaded on the same, or different processors.
- Exception and abort handling.
- Access to the operating system facilities from a high level language.

These facilities were provided by equipping each processing node with a specially modified version of the TRIPOS operating system. In its "standard" form, TRIPOS [6.8-6.10] is an

unprotected, multi-tasking operating system designed for running on a single processor. It consists of a kernel and device drivers which, for speed, are written in assembler, and a number of other utilities such as a file and console handler which are written in the BCPL programming language [6.11, 6.12]. The standard TRIPOS operating system was modified for use with the multi-processor computer hardware, and a detailed description of the modifications is given in Reference 6.1. The major changes which were made are outlined below.

Since the IO processor is the only processor requiring all the operating system functions listed above, two modified versions of TRIPOS were developed, one for the IO processor (referred to as IO-TRIPOS) and the other for the slave processing nodes, (referred to as slave-TRIPOS). In point of fact, two versions of slave-TRIPOS had to be developed, one for the MC68000 processing node and one for the MC68020 processing node. This was because of differences in the hardware design of the respective processing nodes. Essentially slave-TRIPOS does not support the IO management of IO-TRIPOS, which allows its code size to be much smaller.

As has been mentioned, TRIPOS is a multi-tasking operating system, which means that more than one task can be resident on a single processor, although obviously at any instant the processor can only be computing any one of these tasks. The approach adopted in producing a multi-processor TRIPOS, was to extend the TRIPOS multi-tasking capability so that tasks can reside on more than one processor, permitting each processor to calculate a task at the same time. If this is done in such a way that the tasks can be executed

independently of the processor on which it is loaded, it has the significant advantage that multi-processing software can be written in the same way that multi-tasking software is written for a single processor TRIPOS system. The tasks are then computed in parallel rather than serially, and since the tasks are processor independent, it becomes a simple operation to change the processor on which a task is to be loaded and executed.

6.4.1 Communication Between Tasks

In order to permit tasks to be executed by different processors, it was necessary to modify and extend the way in which TRIPOS exchanges information between tasks. The modifications allow information to be exchanged between tasks running on different processors, as well as between tasks running on the same processor. TRIPOS tasks communicate by reference, which means that rather than copying memory from one area to another, they simply pass a pointer relating to the data (a value from which the memory address of the data can be obtained), which the receiving task can then use to access the information. The act of sending a pointer is referred to as packet sending and in a shared memory multi-processing system is an extremely fast method of exchanging information and sychronizing events.

In a single processor TRIPOS system, the operating system allocates a unique identifier to each task when the task is created. When sending a packet, a task uses the identifier to indicate to the operating system what is the destination task for the packet, and

the operating system adds the packet to the destination task work queue. In a multi-processor environment the situation is more complex, since the operating system also needs to know whether, or not, the destination task is loaded on the same processor as the sending task. This problem was solved elegantly by using a two part task identifier, one part containing the number of the processor on which the task is loaded, and the other part containing the task number, which is unique to the processor on which it is loaded. The processor number is also unique and depends upon the location of the processor local memory in the global address space. When a packet is sent by a task, the operating system determines whether or not the destination task and the sending task are loaded on the same processor. If they are, the operating system adds the packet to the destination task work queue as before. If, however, the task is loaded on a different processor, the appropriate processor is notified (by interrupting it) and its operating system is instructed to add the packet to the destination task work queue. Thus, by re-writing the packet handling routines used by TRIPOS in this way, tasks being executed by different processors are able to exchange information.

6.4.2 Processing Node Server Task

Each slave node runs a task called a serving task, which indirectly provides the means for tasks running on other processors to access the slave node local operating system routines. These local operating system routines perform functions such as task and memory management.

By way of illustrating how the serving task mechanism operates, consider the simple case of an application task executing on processor P_1, needing to have some memory allocated for its use on processor P_2. Since each processor is responsible for managing its own local memory, before the application task can use the P_2 memory it must by some means first ask the P_2 operating system to allocate to it the required space. The application task itself cannot request the memory, since it has no means of communicating with the P_2 operating system. However, since it can communicate with the P_2 server task, it sends the server task a request for the memory (by sending it an appropriate packet). In response, the server task asks the P_2 operating system to allocate the required memory and then returns the result to the application task loaded on P_1. It should be noted that the operating system is not actually aware that it has allocated memory for a task which is running on another processor.

Other requirements, which the server task has to support, are listed below:

- Task management; for example creating or deleting a task, changing the priority of a task, holding or continuing with a task etc.
- Memory management: allocation and deallocation of memory.
- IO management: overseeing device driver functions.
- Library management: installation or deletion of software libraries, for example a library to perform floating point

calculations.

6.4.3 IO Processor Server Task

A task running on a slave processor can request the IO processor to perform certain IO operations on its behalf, for example, to print a message at the console. To do this, the IO processor runs a server task similar in principle to the server task running on the slave processing nodes. The IO server functions are:

- Console output: Output messages and parameter values at the console.
- Clock: Provide access to the clock device on the IO processor.
- Memory management: Memory management of its local memory space and the backplane memory card.

6.4.4 The Multi-Processor Debug System

The single processor TRIPOS operating system provides a powerful debug system for monitoring and controlling the execution of code. Typical functions provided are: setting breakpoints, examining and changing task variables, single stepping program code and disassembling code. A debug system was developed to support similar functions on the multi-processor hardware. This proved to be invaluable while debugging the engine model software, - indeed without the debug system, getting the engine model software to run would have been a truly daunting task.

The most useful feature of the debug system is its ability to set breakpoints at appropriate points in the code, on any processor. When a breakpoint is encountered, program execution is halted, and the state of the program at that instant is frozen. The debug system can then be used, for example, to inspect model variables and examine the run-time stack which is used (by high level languages such as BCPL) for the storage of local variables, for passing parameter values during procedural calls, as well as storing their return addresses. Thus, by using the breakpoint facility to control and examine the execution of a task, it is possible to identify a software bug.

The debug system is always loaded on the computer system and can be entered automatically, or by operator choice. It is entered automatically whenever a run-time error occurs on any processor (such as encountering a breakpoint). Each slave processor runs a debug handler, which reports local run time errors to the main debug system residing on the IO processor. The main debug system then informs the user (by displaying a message on the console) that a run time error has occurred and gives some information about the type of error and the processor on which it occurred. Having sent its debug message to the IO processor, the debug handler on the slave processor then halts the processor. The main debug system (running on the IO processor) can then be used to examine and change the slave processor local memory, as instructed by the user entering commands at the console.

The debug system can also be entered by the user at the console. Using debug commands, the user can select the processor

and task (on that processor) to be examined, even while the task continues to operate.

6.4.5 Multi-Processor Commands

A number of commands were written for the multi-processor system and are listed in Reference 6.1. The commands carry out functions such as reading the slave TRIPOS operating system images from disc, down loading them to a specified processor and starting the processor running.

6.4.6 The BCPL Programming Language [6.11,6.12]

A description of the computer system software would not be complete without a description of the BCPL programming language in which the diesel engine simulator and much of TRIPOS is coded. BCPL is a modular, block-structured high level language and provides many program flow constructs. It was designed to be a systems programming language and has been widely used in some academic and research institutions.

BCPL is compiled to an intermediate code (referred to as o-code) which is then interpreted, or passed through a second stage, (the so called "code generator") to produce machine code. Two code generators were used on the multi-processor system, one to produce code for the MC68000 processor and the other for the MC68020 processor. Although the MC68020 processor is object code compatible with the MC68000 processor, the MC68020 has several new and enhanced

instructions which allow certain operations to be coded more efficiently, as well as providing direct software support for the MC68881 co-processor. The existing MC68000 code generator was modified to take advantage of these extra instructions when generating code to run on a MC68020 processor.

Unlike most other high level languages, BCPL is a "typeless" language which means that it recognises no distinction between types of data, and is only concerned with fixed length bit patterns (32 bits in the case of the language implementation used). Thus, whereas a programmer conceptually makes a distinction between data types such as integer and floating point numbers, BCPL makes no distinction (‡). The programmer must therefore be extra vigilant to ensure that data types are not mixed accidently, for example, multiplying an integer and floating point value, since the BCPL compiler will quite happily compile the code even though the result may be meaningless.

The advantage of a typeless language is that the programmer is free to choose the data structures which are best suited to the application, rather than having to accept any inherent limitations which may be imposed on the data structures in a strongly typed language. This gives the programmer complete freedom of choice and indeed the basic philosophy of the BCPL language is that the

---

(‡) Typed languages enforce a distinction by associating variable names with types of data, for example the FORTRAN language assumes that variables which begin with the characters, I,J,K,L,M and N are integer, unless explicitly told otherwise.

programmer knows best.

6.4.7 Booting the Multi-Processor Operating System

Finally, the sequence of events which load the operating system when the computer is switched on, (or reset) is as follows. The bootstrap program which resides in EPROM on the IO processor board, loads the IO processor with the IO-TRIPOS operating system and starts it running. The user can then use the computer system as a "standard", single processor TRIPOS system. To use the computer system in its multi-processor mode, commands are entered by the operator which cause the IO processor to load the slave-TRIPOS operating system from disc, and download it to each specified processing node (using the system bus). The processor then starts running and an application task running on the IO processor can then create a task(s) to be executed by the slave processing node(s).

6.5 Summary

This chapter has described the choice of a tightly coupled MIMD type multi-processor computer system for computing the filling and emptying engine model in parallel. The computer system hardware employs a number of processing nodes based on the MC68000 or MC68020 microprocessor, with local memory and optional IO features being provided on the processing nodes. The MC68020 processing node is also equipped with additional hardware to perform binary floating point operations.

The multi-processor computer operating system was developed from the single processor TRIPOS operating system, with which maximum compatibility was maintained to allow multi-processor software to be written in the same manner, as multi-tasking software is written for a single processor TRIPOS system, and to allow the use of the usual TRIPOS program development utilities. A powerful centralised debug system has been developed to allow tasks resident on any processor to be examined, either following a run time error, which is reported to the operator at the console, or during normal systems operation.

6.6 References

6.1 L A Dale.
Real time modelling of multi-machine power systems.
1986 PhD Thesis, University of Bath.

6.2 MC68000, 16/32 -bit Microprocessor Programmers Reference Manual, fourth edition.
1984 Motorola.

6.3 D G Tanner.
Real time simulations of power systems.
1982 PhD Thesis, University of Bath.

6.4 S K Williams.
Power system optimisation and stability studies using Real-time simulation.
1986 PhD Thesis, University of Bath.

6.5 MC68020 32-bit microprocessor users manual.
1984 Motorola.

6.6 MC68881 Floating point co-processor users manual, 1st edition.
1985 Motorola.

6.7 J Coonen et al
A Proposed Standard for Binary Floating Point Arithmetic.
Oct 1979, ACM Signum Newsletter.

6.8 M Richards, A R Alyward, P Bond, R D Evans, B J Knight.
TRIPOS - a portable operating system for mini-computers.
1979 software practice and experience, vol 9, p513-526.

6.9 T J King.
TRIPOS user guide, Issue 2.
March 1983, School of Mathematics, University of Bath.

6.10 T J King.
TRIPOS technical manual, Issue 2.
May 1983, School of Mathematics, University of Bath.

6.11 M Richards, C Whitby-Stevens.
BCPL the Language and its Compiler.
1982 Cambridge University Press.

6.12 T J King.
TRIPOS programming guide, Issue 2.
April 1983, School of Mathematics, University of Bath.

6.7 Tables

MC68881 Monadic Operations

FABS	Absolute Value
FACOS	Arc Cosine
FASIN	Arc Sine
FATAN	Arc Tangent
FATANH	Hyperbolic Arc Tangent
FCOS	Cosine
FCOSH	Hyperbolic Cosine
FETOX	e to the power x
FETOXM1	e to the power x-1
FLOG10	Log Base 10
FLOG2	Log Base 2
FLOGN	Log Base e
FLOGNP1	Log Base e of (x+1)
FNEG	Negate
FSIN	Sine
FSINCOS	Simultaneous Sine and Cosine
FSINH	Hyperbolic Sine
FSQRT	Square Root
FTAN	Tangent
FTANH	Hyperbolic Tangent
FTENTOX	10 to the Power x
FTWOTOX	2 to the Power x

MC68881 Dyadic Operations

FADD	Add
FCMP	Compare
FDIV	Divide
FMOD	Modulo Remainder
FMUL	Multiply
FSGLDIV	Single Precision Divide
FSGLMUL	Single Precision Multiply
FSUB	Subtract

Table 6.1 MC68881 Floating Point Co-Processor Operations.

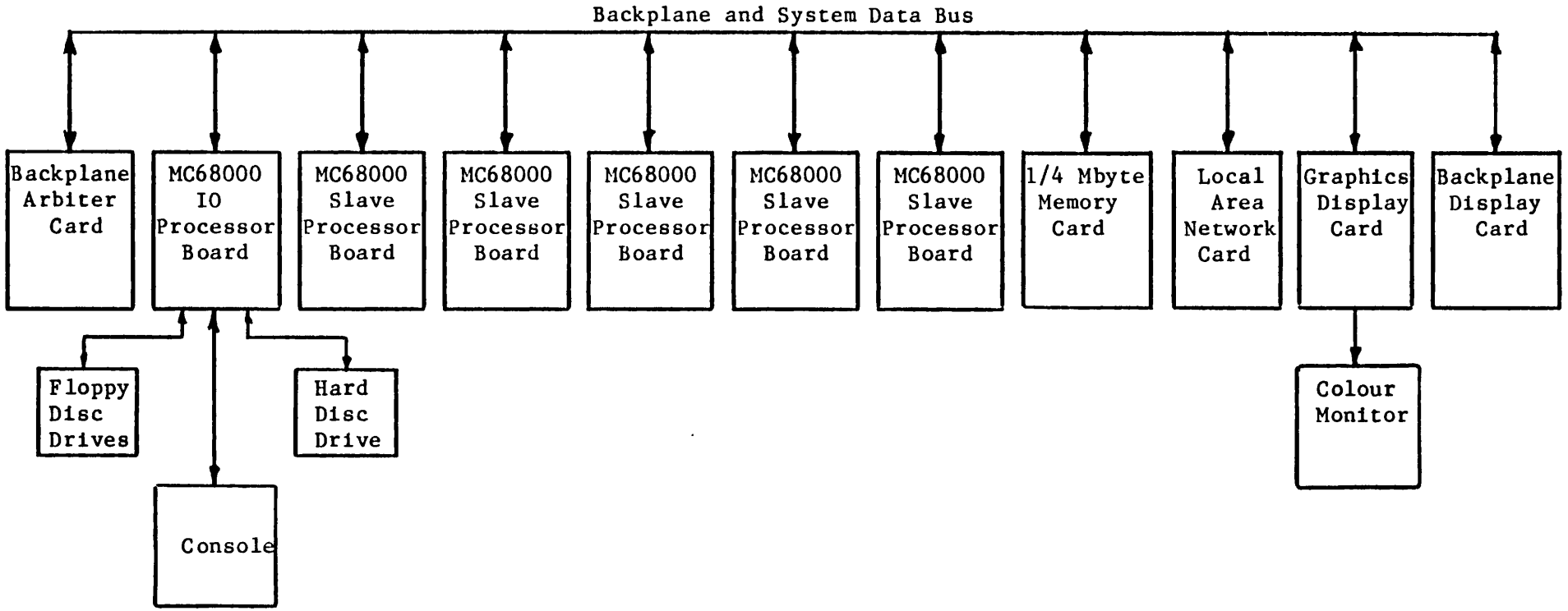


Figure 6.1 Schematic of the MC68000 Based Parallel Computer System.

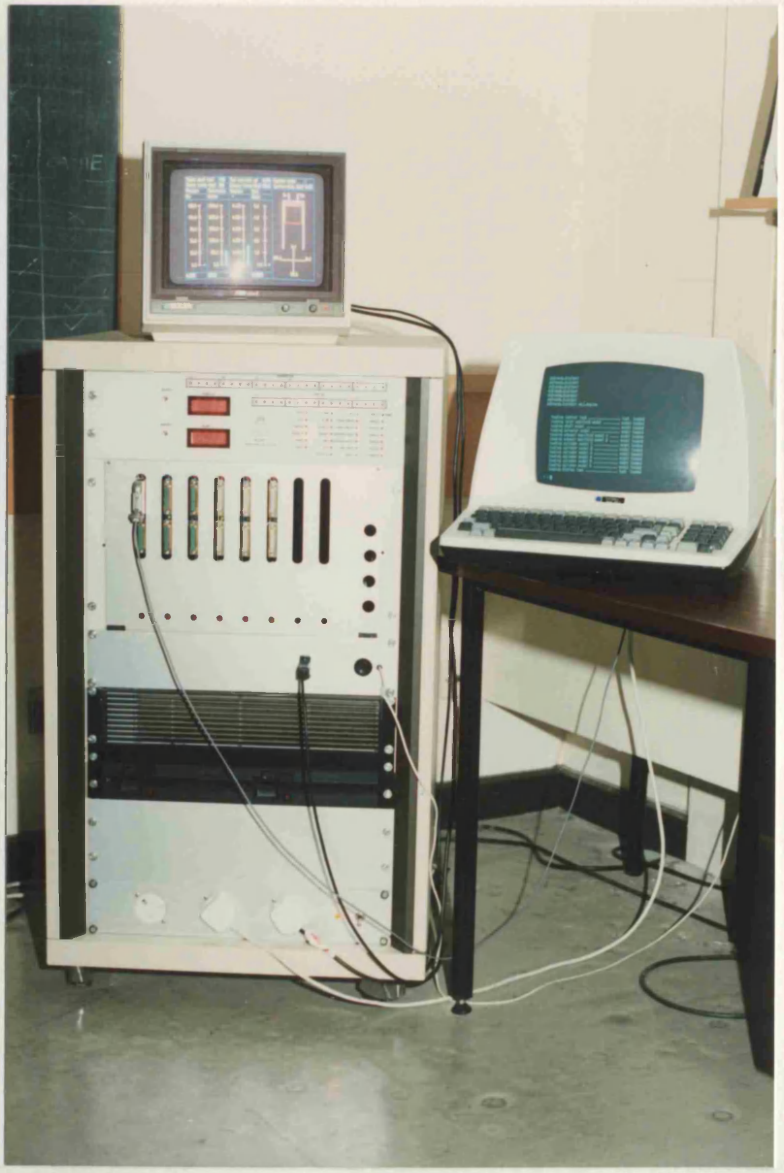


Figure 6.2 Parallel Computer System.

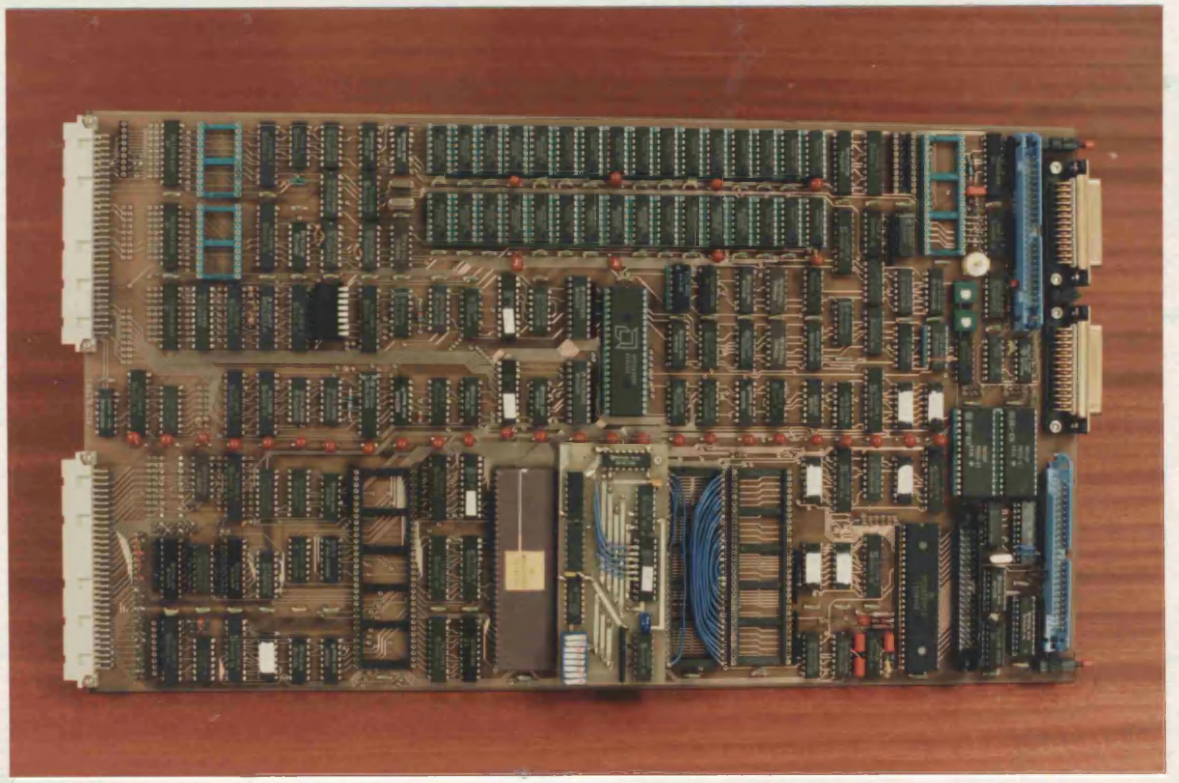


Figure 6.3 MC68000 Based Processing Node.

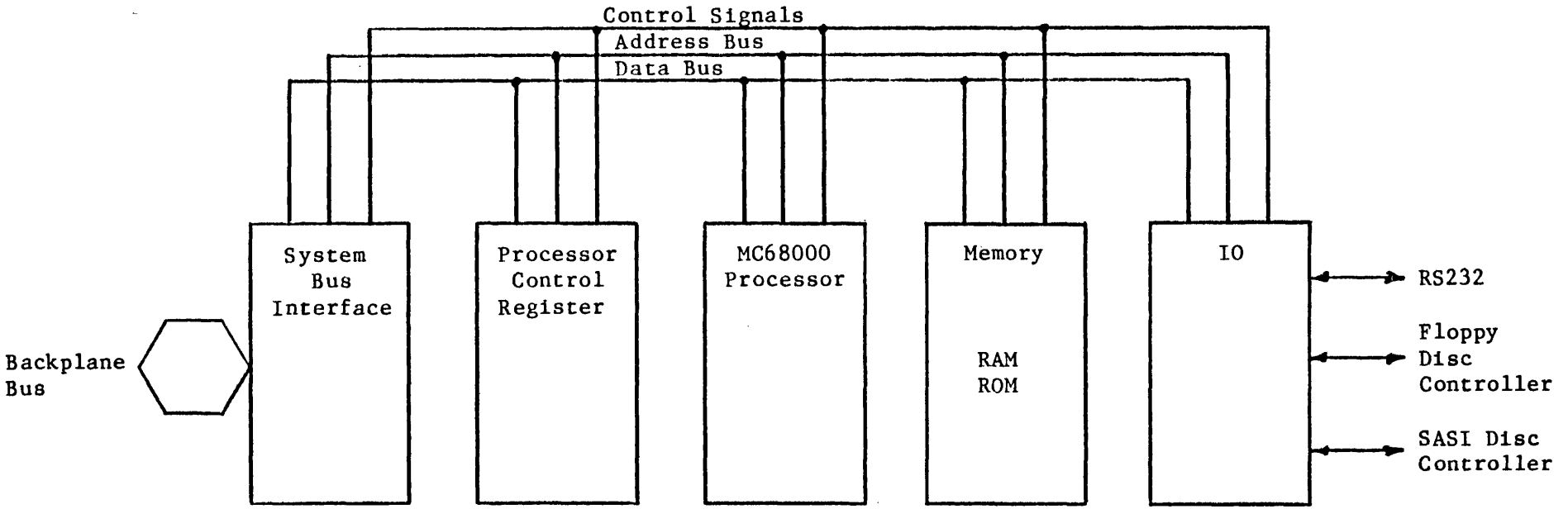


Figure 6.4 Block Diagram of the MC68000 Based Processing Node.

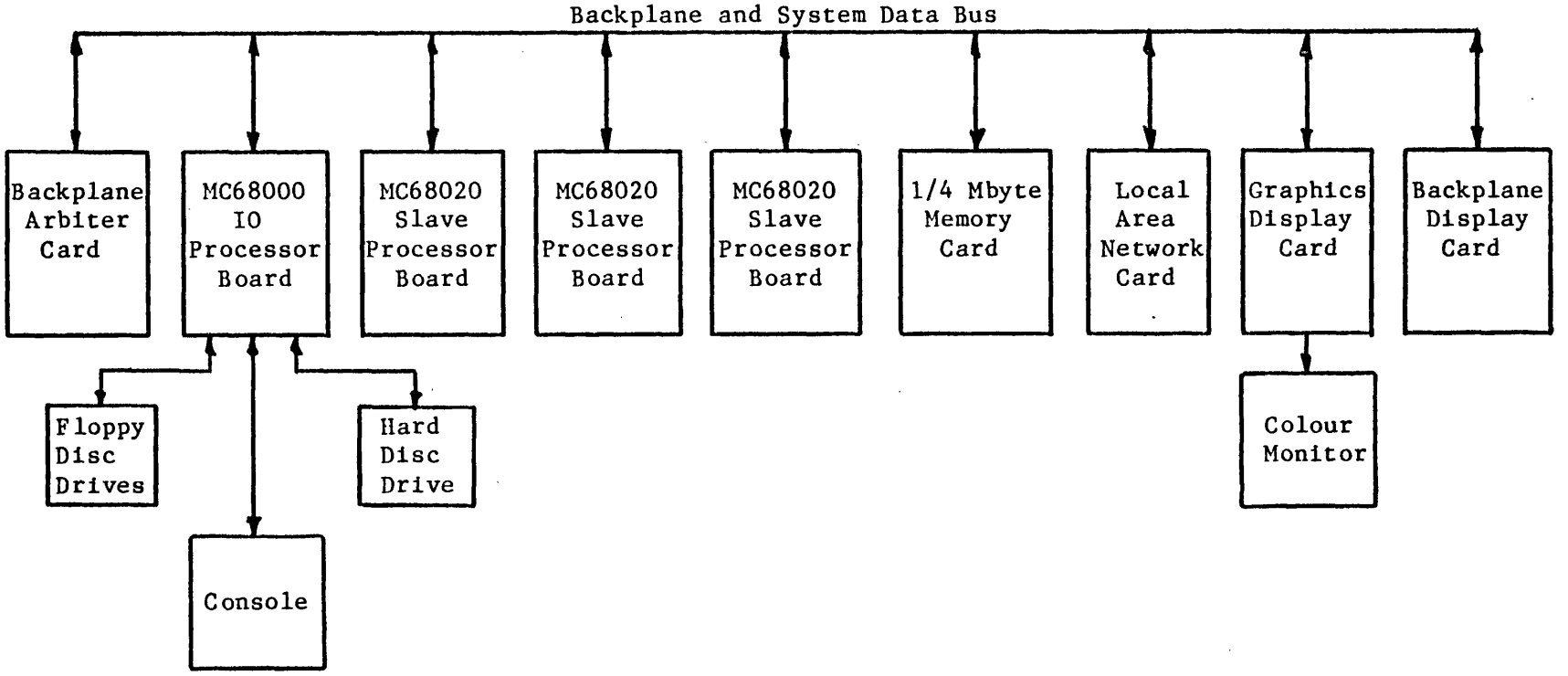


Figure 6.5 Schematic of the MC68020 Based Parallel Computer System.

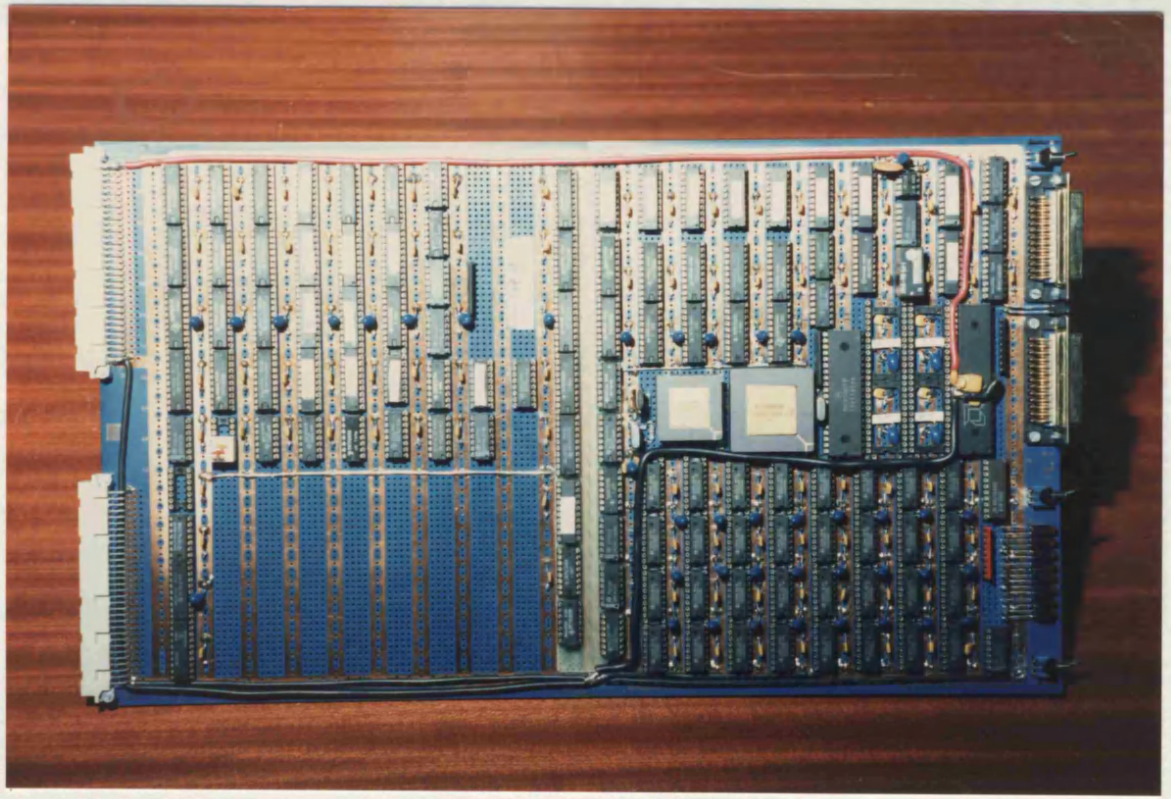


Figure 6.6 MC68020 Based Processing Node.

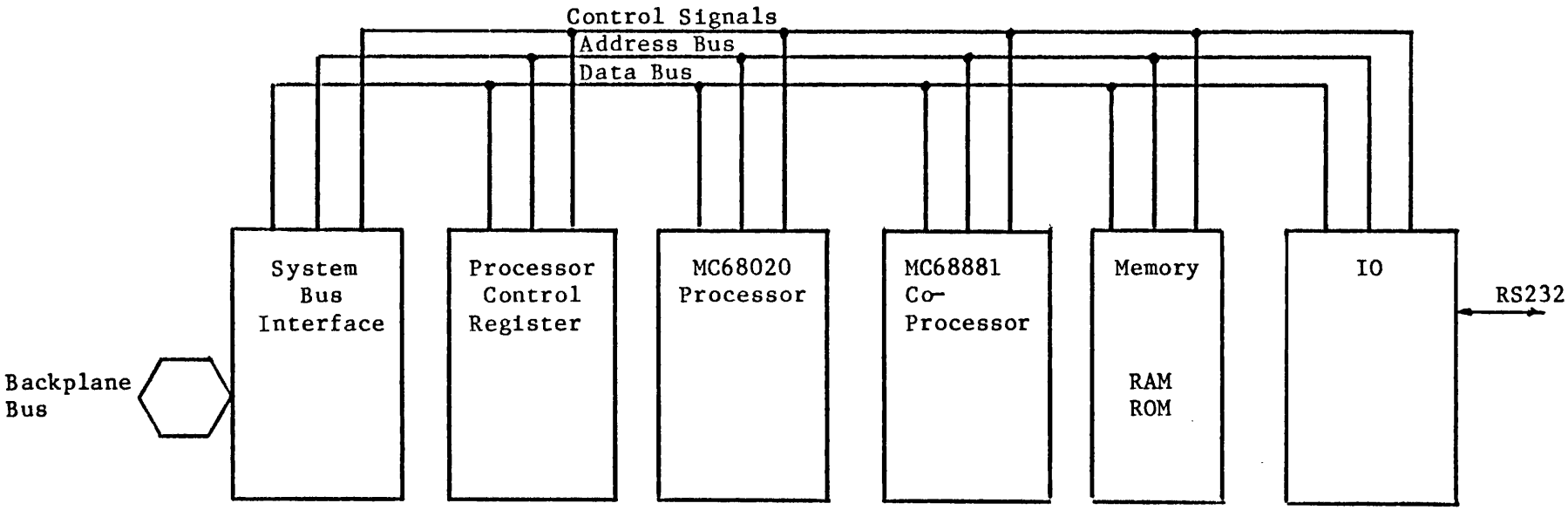


Figure 6.7 Block Diagram of the MC68020 Based Processing Node.

CHAPTER 7

7.1 Engine Simulator Software

This chapter describes the software which has been developed to compute the filling and emptying engine model described in Chapter 4: the software implements the solution procedure described in Chapter 5, using the multi-processor computer system described in Chapter 6.

The principle software tasks, and the information which has to be exchanged between tasks are determined by the way in which the engine model was divided for computing in parallel, and by the method adopted for solving the engine model equations. In addition, other tasks are required to provide simulator facilities. The following software tasks can be readily identified:

- Calculation of the behaviour of cylinder and manifold control volumes, the control actuators and the engine shafts.
- Overall supervision of the engine model solution procedure.
- Generation of a moving display of engine model performance on the colour graphics monitor.
- Provision of general simulator facilities, such as the recording and plotting of engine model responses on the graphics monitor, as and when the operator enters the appropriate command at the console.

A block diagram of the simulator software tasks is shown in Figure 7.1. It was decided to keep the display and command tasks separate from the tasks which are directly responsible for computing the engine model equations, so that the display and command tasks can be executed by the IO processor, leaving the supervisor and engine model tasks to be executed by the slave processing nodes. This arrangement has the advantage that the engine model can continue to operate concurrently with operation of the display and command task, and thereby minimises the effect that the use of the display and simulator facilities has on the execution speed of the engine model. The command and display tasks always reside on the IO processor, but the other simulator software tasks were written as independent operating system tasks which can be loaded on to any processing node for execution.

A high level programming language was used to code the tasks, rather than an assembler, since this makes program development much quicker and the code easier to understand. A primary requirement for the programming language is that it should have a high degree of compatibility with the operating system, and for this reason BCPL was adopted, since as has been mentioned, BCPL is the native language of the operating system.

A single precision floating point data format (IEEE [7.1]) was considered to be an acceptable compromise between execution speed and accuracy, and was used to represent all engine model variables.

7.2 Simulator Task Communication

Simulator tasks use packets to exchange information and maintain task synchronisation and a high proportion of each task code is concerned with the management of these exchanges. Because of the importance of the exchanges, a detailed description of packet handling is given before a description of the individual simulator tasks (in Section 7.3) since, with minor variations, the method used to handle packets is common to all tasks. The description is given from the point of view of the simulator tasks rather than the operating system. The only concern of the operating system is to ensure that each packet sent by the tasks arrives at the correct destination task, and is subsequently returned to the sender. It is not in any way concerned with interpreting what action is required as a result of a packet being sent; this is the sole concern of the sending and destination tasks.

Simulator tasks which exchange information are shown linked together on the block diagram of the simulator software shown in Figure 7.1. It can be seen that simulator communication is hierarchical in nature, i.e all exchanges originate with a task sending a packet (or packets) to a task (or tasks) at the same level or at a lower level than itself.

The exchanges between the supervisor task and engine model tasks occur very frequently. In almost all these exchanges, the supervisor task sends packets to the engine model tasks to evaluate and integrate their state equations, in order to advance the engine model solution, as described in Chapter 5. The exchanges between

the display task and engine model tasks also occur frequently and consist of the display task sending packets to the engine model tasks, to return engine model performance for subsequent display by the display task on the graphics monitor. The exchanges which take place between the command task and supervisor task, and between the command task and engine model tasks are of infrequent occurrence, and only arise when the operator enters a command at the console. These exchanges handle activities such as plotting or storing engine model responses, updating the engine model with new control settings and, of course, stopping the simulation. Finally, the exchanges which occur between the command task and display task allow the command task to halt the display task (and subsequently to continue it), so that the command task itself can use the monitor for graph plotting.

7.2.1 Packet Handling

The standard data structure of a packet is shown in Figure 7.2. It consists of eleven entries, - the link entry, destination task entry, type entry, two result entries and six argument entries. Before a packet can be sent, the sending task is responsible for filling appropriate entries which are described below.

The link entry (which is always set to the constant value -1, by the originator) is reserved by the operating system for its own purposes. The destination entry is set to the task number of the task to which the packet is to be sent. As was explained in the

previous chapter, the task number is a unique identifier which enables the operating system to distinguish between every task on the computer system. The type entry stores a unique identifier which is used by the sending task to instruct the destination task which operation it is to perform. For example, in the engine simulator, a value 10 is used to instruct the control volume tasks to set up initial conditions, a value 32 to perform predictor calculations, a value 33 to perform corrector calculations and a value 31 to update model variables etc.

The destination task usually requires additional information to carry out the specified operation and this is provided by the six argument entries, in the packet. The sending task sets these with any required information prior to sending the packet. For example, when a task sends a command to a control volume task, to perform a predictor calculation, it might use entry "pkt.argl" to specify the integration step size to be used.

Once the destination task has carried out the operation specified by the packet, it may have to return results to the sending task, and this is the purpose of the two result entries in the packet. The destination task fills the result entries as required, and then returns the packet to the sending task.

The packet data structure contains entries for six arguments and two results, but for many data exchanges this is inadequate. The problem is easily overcome by using an argument and/or result entry to hold a pointer which identifies the start of another block of memory which can contain additional information. It is, of

course, important to ensure that during all exchanges, the sending and destination task both use the same "type" codes, and that information is transferred using the same argument and result entries.

Figure 7.3 illustrates the basic program code required to manage a packet transfer between a sending and destination tasks. The exchange commences at "A" with the sending task setting up the packet structures and queuing them to the destination tasks using the TRIPOS "QPTK" (queue packet) primitive. As each packet is queued, the sending task marks the packet as having been sent, so that it can subsequently mark the packets back, as they are returned. Once all the packets have been queued, the sending task calls the TRIPOS primitive "TASKWAIT", at "B" to wait for the packets to be returned; having called "TASKWAIT", the operating system suspends the sending task until a packet is returned.

The destination tasks (which can be running on the same or different processors to that running the sending task) are made ready to run by the arrival of the packet on their work queue at "C". They immediately inspect the packet "type" entry and jump to the appropriate section of code which performs the required operation. The tasks obtain any additional information which they require from the argument entries in the packet, and perform the required calculations; they then set up the packet result entries (if required) in readiness for returning the packet. In general, the last operation which the tasks perform (prior to calling "TASKWAIT" at "C") is to return their packet to the sending task using the TRIPOS "RETURNPKT" primitive at "D". One exception to

this is when a task receives a packet instructing it to delete itself. Clearly, the task must return its "delete" packet before deleting itself, otherwise the task will not then exist to return the packet, as is shown at "E" in Figure 7.3.

The sending task becomes ready to run (again), when a returned packet arrives on its work queue. Having received a returned packet, it calls the routine "handle.packet" which inspects the packet type entry and jumps to the appropriate section of code which marks the packet as returned, thus indicating that its action has been completed. The packets can be returned in any order, and once they have all been returned, the task is ready to perform new operations at "F". It should be noted that the method of queuing packets to tasks (using QPKT) to perform some operation and waiting for the packets to be returned (using TASKWAIT), enables the task sending the packets to keep the tasks synchronised, as well as exchanging information between the tasks.

The description given above is considerably simplified, in that it assumes that tasks can only send or receive packets. In practice some tasks (ie the supervisor and display task) both send and receive packets and an important function of their "handle.packet" routine is to sort newly arrived and returned packets. The routine has to decide whether a packet which has just arrived on its work queue is a packet which it sent earlier (in which case it marks the packet as returned), or whether it is a packet from another task, instructing it to perform some operation. If the packet is of the second type, the task either deals with it immediately, or marks it as having arrived, to be dealt with as soon

as possible. If, for example, a task receives a packet to delete itself, it must first wait for all its outstanding packets to be returned before it deletes itself, otherwise the computer system will be left with packets having no task to return to.

7.3 Description of the Simulator Tasks

7.3.1 Command Task

The engine simulator is started by entering its name (obj.engine) as a normal command at the console; this causes the operating system to load the command task to the IO processor and start it running. Pseudo-code for the command task is shown in Figure 7.4. This shows that the first operation performed by the command task is to load the other simulator tasks (ie display task, supervisor task, control volume tasks, control actuator task and shaft tasks) from disc to the specified processing nodes and to send them any initial conditions they may require, including initial values for the engine model variables. Once initialisation is complete, the engine simulator is ready to run and the command task sends a packet to the supervisor task to start the engine model calculations, and a packet to the display task to start displaying engine model performance on the graphics monitor: the command task then displays a prompt at the console and waits for the operator to enter a command. A description of the commands available to the operator is given in Appendix A3; they include such operations as plotting engine model responses on the monitor, saving engine model responses to a file on disc, and changing the engine control

settings.

One of the more useful facilities which the command task provides is to plot engine model responses on the graphics monitor. The command task uses the Graphics Kernel System (GKS) [7.2] to drive the efcis card and photographs of typical responses are shown in Figures 7.5a to 7.5f. Figure 7.5a and 7.5b show how the gas pressure and temperature in a cylinder control volume model vary over a complete engine power cycle. Figure 7.5c and 7.5d also show the variation of gas pressure and temperature, but this time for six cylinders and over a period of three power cycles. Finally, Figure 7.5e and 7.5f show the way in which turbocharger speed and boost pressure change over a 24 second period, when step changes are made to fuel rack position.

On line help (ie the "HELP" command) is available to assist the operator to use the simulator, by providing a brief description of the simulator commands and facilities. Additionally, all simulator commands use the standard TRIPOS argument parser; if the command name is entered followed by a question mark, the parser will display the command arguments at the console.

It should be stressed that execution of commands requested by the operator has a negligible effect on the execution speed of the engine model. As has already been explained, this is because the slave processing nodes which are responsible for computing the engine model tasks, can continue to operate concurrently with commands being executed by the IO processor.

7.3.2 Supervisor Task

The primary responsibilities of the supervisor task are to manage the solution procedure of the engine model and to arrange for engine responses to be recorded at regular intervals of crankshaft rotation. Pseudo-code for the supervisor task is given in Figure 7.6.

The routine "solve.model" (Figure 7.6) is responsible for managing the engine model solution procedure described in Chapter 5. This routine involves sending packets to the engine model tasks to repeatedly evaluate and integrate their state equations, until a satisfactory solution of the equations is achieved in all the control volumes. The crankshaft angle is then incremented by a small step and the solution procedure started again.

The supervisor task is also responsible for ensuring that engine responses are recorded at regular intervals of crankshaft rotation. The supervisor task sends the control volume tasks a command to record their gas responses every four degrees of crankshaft rotation, and the supervisor task itself records slowly changing engine responses (listed in Table 7.1), at much larger intervals of crankshaft position.

In addition to these primary responsibilities, the supervisor task also performs a variety of miscellaneous operations for the command task (when sent the appropriate packet), and these are listed in Table 7.2.

7.3.3 Control Volume Task

There are two basic types of control volume task which represent the behaviour of a cylinder and manifold. There are also two versions of the manifold control volume, one representing an inlet manifold and the other an exhaust manifold. As is to be expected, the structure of the control volume tasks, and the actions they perform are similar, the only difference being that evaluation of the state equations is different for a cylinder and manifold, as was discussed in Chapter 4.

Pseudo-code for a cylinder control volume task is given in Figure 7.7. The more important operations which the control volume task performs (when it receives the appropriate packets) are described below:

- Initialization packet (act.init): The control volume tasks receive an initialization packet from the command task, prior to the engine model solution procedure commencing. The packet specifies initial values for the variables listed in Table 7.3a for a cylinder control volume, and in Table 7.3b for a manifold control volume. Using the initial conditions, the control volume tasks calculate the state equations to obtain the rates of change of the gas states at the specified crankshaft position. Finally, the task returns the initialization packet, so that the command task knows that the control volume task is initialized and ready to commence solution of the state equations.

- Predictor packet (act.predict): During solution of the engine model, the supervisor task regularly sends a predictor packet to the control volume tasks (see Chapter 5). This instructs the control volumes to predict the gas state at a new crankshaft position θ_{n+1} using the known gas conditions at the current crankshaft position θ_n. On receiving the predictor packet, the control volume task applies the predictor formula (5.1) to the gas rates at crankshaft position θ_n to obtain predicted estimates of the gas states at crankshaft position θ_{n+1}. The integration step size for use during the calculations is supplied with the packet. The control volume then calculates the gas properties and gas pressure at the new crankshaft position θ_{n+1} (and geometry and combustion, if the control volume represents a cylinder). Finally, the control volume returns the packet together with those variables which are required by connected control volume tasks as boundary conditions for the next stage of the engine solution procedure (ie corrector calculations); these variables are listed in Table 7.4a and 7.4b for a cylinder and manifold control volume respectively.

- Corrector packet (act.corrector): When all the predictor packets have been returned by the engine model tasks, the supervisor task sends the control volume tasks a corrector packet (as described in Chapter 5). This packet instructs the control volume tasks to obtain corrected estimates of the gas conditions at the new crankshaft position θ_{n+1}. On receiving the corrector packet, the control volume tasks calculate the

rate of heat transfer between the gas and gas exposed walls, rate of mass flow entering (or leaving) the volume and the state equations at crankshaft position θ_{n+1}. The boundary conditions necessary for performing the calculations are supplied by the supervisor task when it sends the packet and are listed in Tables 7.5a and 7.5b for a cylinder and manifold control volume respectively. Having evaluated the state equations, the gas rates are integrated using the corrector formula (5.2) to obtain improved estimates of the gas states at θ_{n+1} and the differences between these and the previous state estimates (at θ_{n+1}) are compared against an allowed tolerance. If the difference for any state exceeds the permitted tolerance, a stability flag is set to indicate a failed stability test. The control volume tasks calculate the gas properties and gas pressure (using the most recent corrected state estimates), for use as boundary conditions by connected control volume tasks if the corrector calculations have to be repeated (as described in Chapter 5). Finally, the control volume tasks return the corrector packet, together with the results of the calculations and the stability flag. Tables 7.4a and 7.4b list the variables which are returned by a cylinder and manifold control volume respectively.

- Update packet (act update): Once a satisfactory solution of the state equations is achieved in all the control volumes, the supervisor task sends all the engine model tasks an "update packet". This causes the control volumes to update the current values of the state variables to the new corrected

values appropriate for the next crankshaft position (θ_{n+1}) in preparation for advancing the engine model solution by another small step in crankshaft angle. Finally, the control volume tasks return the update packet.

- o **Miscellaneous packets:** The control volume tasks also perform a variety of other operations when sent an appropriate packet and these are listed in Table 7.6. The operations are not directly associated with solution of the control volume state equations but relate to activities such as recording and displaying control volume gas responses, the display task etc.

7.3.4 Engine Shaft Task

The shaft task evaluates and integrates the state equations which represent the angular acceleration of the engine crankshaft (4.62) and turbocharger shaft (4.48). It does this when the supervisor task sends a calculate packet "act.calc" (at the same time as it sends the control volume tasks a predictor packet). The calculate packet includes data for engine brake torque, turbine torque, compressor torque and the integration step size, all of which are required to perform the task. On completing the calculations, the shaft task returns the packet together with the new value of engine and turbocharger speed. As with the control volume tasks, the shaft task updates the engine and turbocharger speed to the new values (at θ_{n+1}) on receipt of an update packet from the supervisor task.

The shaft task also performs various other operations (mostly

for the command task); which are listed in Table 7.7. Pseudo-code for the shaft task is given in Figure 7.8.

7.3.5 Actuator Task

The actuator task evaluates and integrates the state equations (4.58) which represent the dynamic response of the control actuators (fuel rack, fuel injection timing and turbine variable geometry actuator) and calculates fuel delivery. The actuator task performs these calculations on receiving a calculate packet "act.calc" from the supervisor task at the same time as the shaft task is sent its calculate packet. The packet includes the integration step size to use, engine speed and the demanded control actuator settings. On completing the calculations, the actuator task returns the packet together with information on the fuel shot mass, static timing and turbine mass flow restriction. The actuator task also updates its actuator position to the new actuator position (at θ_{n+1}) when it receives an update packet from the supervisor task.

The actuator task performs various other operations when sent an appropriate packet (primarily for the command task) and these are listed in Table 7.8. Pseudo-code for the actuator task is given in Figure 7.9.

7.3.6 Display Task

The display task is responsible for generating a moving

display of engine performance on the graphics monitor which is run as a background task on the IO processor. A photograph of the display is shown in Figure 7.10. The display shows the performance of a cylinder control volume; it gives a digital read out of engine speed, previous fuel shot mass, static and dynamic fuel injection timing, ignition delay, pressure, temperature, fuel-air ratio and mass of gas in the cylinder. The pressure, temperature, fuel-air ratio and mass of gas are also displayed as moving bar charts. Finally, on the right of the display a schematic of a cylinder is shown; this shows the engine valves opening and closing, the piston moving up and down and the crankshaft rotating in synchronisation with the cylinder control volume calculations.

As soon as the engine model calculations are running, the command task sends the display task a "start display" packet. The packet specifies which cylinder control volume is to be interrogated to obtain the information required to generate the display. The display task then sends this cylinder control volume task a packet at regular intervals, requesting it to return the necessary performance data. The information returned by the cylinder is listed in Table 7.9. The display task updates the graphics display when the data packet is returned to it. Unlike the command task, the display task does not use the Graphics Kernel System (GKS) to control the graphics, because GKS involves so much computation that the display task would be unable to update the display quickly enough to generate a smooth screen movement. Instead, the display task controls the graphics card directly by sending its driver packets to re-draw the moving picture elements.

In addition to its primary purpose of generating the graphics display, the display task performs other operations for the command task. These allow the command task to change the cylinder control volume being displayed, stop and start the display task (so that the command task itself can use the graphics card for plotting engine responses) and delete the display task. Pseudo-code for the display task is given in Figure 7.11.

7.4 Summary

A description has been given of the software developed to implement a diesel engine simulator based on solving the filling and emptying diesel engine model in parallel. The software tasks required to perform engine model calculations and provide simulator facilities have been presented together with the method used by the tasks to exchange information and maintain task synchronisation, using packets.

By making the IO processor responsible for providing simulator facilities and the slave processing nodes responsible for performing engine model calculations, it has been possible to provide flexible and powerful simulator capabilities without imposing any significant delay in carrying out the engine model calculations. The simulator allows the operator to change engine control inputs, record, save and display engine model responses at any time. When the graphics monitor is not being used to display engine model responses, it is used to provide an animated display of a cylinder control volume performance. All simulator facilities are controlled by the operator entering commands at the console.

7.5 References

7.1 J Coonen et al.
A Proposed Standard for Binary Floating Point Arithmetic.
Oct 1979, ACM Signum Newsletter.

7.2 Hopgood S R A, Duce D A, Gallop J R, Sutcliff D C.
Introduction to Graphics kernel Systems (GKS).
1983, Academic Press.

7.6 Tables

Symbol	Name of Engine Variable
ω_e	Engine Speed
T_b	Engine Brake Torque
ω_{tc}	Turbocharger Speed
$\dot{\omega}_{tc}$	Angular Acceleration of Turbocharger
m_f	Fuel Mass per Injection
v_g	Turbine Variable Geometry Restriction
θ_s	Static Fuel Injection Timing
P_{im}	Inlet Manifold Pressure
T_{im}	Inlet Manifold Temperature
P_{em}	Exhaust Manifold Pressure
T_{em}	Exhaust Manifold Temperature
f_{em}	Exhaust Manifold Fuel/Air Ratio

Table 7.1 Engine Model Variables Recorded by Engine Simulator

Packet Action Type	Operation
act.selfe	Stop the engine model tasks
act.send	Change demanded engine control settings
act.plotbuf	Return the address of the engine time response, data logging buffer
act.logzero	Clear the engine time response data logging buffer

Table 7.2 Operations Performed by the Supervisor Task on the Behalf of the Command Task.

Symbol	Name of Variable
ω_e	Engine Speed
θ	Crankshaft Position (Relative to Cylinder 1)
P	Gas Pressure
T	Gas Temperature
f	Gas Fuel/Air Ratio

a) Cylinder Control Volume

Symbol	Name of Variable
P	Gas Pressure
T	Gas Temperature
f	Gas Fuel/Air Ratio
ω_{tc}	Turbocharger Speed

b) Manifold Control Volume

Table 7.3 Variables Supplied to Cylinder and Manifold Control Volume Tasks during Model Initialization.

Symbol	Name of Variable
P	Cylinder Gas Pressure
T	Cylinder Gas Temperature
f	Cylinder Gas Fuel/Air Ratio
m	Cylinder Gas Mass
R	Cylinder Gas Constant
γ	Ratio of Specific Heats
ho	Specific Stagnation Enthalpy
A_{iv}	Inlet Valve Effective Flow Area
A_{ev}	Exhaust Valve Effective Flow Area
T_i	Indicated Torque

Table7.4a Variables Returned by a Cylinder Control Volume Task, on Completion of Predictor and Corrector Calculations.

Symbol	Name of Variable
P	Manifold Gas Pressure
T	Manifold Gas Temperature
f	Manifold Gas Fuel/Air Ratio
m	Manifold Gas Mass
R	Manifold Gas Constant
γ	Ratio of Specific Heats
ho	Specific Stagnation Enthalpy
T	Compressor (inlet manifold) or Turbine Torque (exhaust manifold)

Table 7.4b Variables Returned by a Manifold Control Volume Task, on Completion of Predictor and Corrector Calculations.

Symbol	Name of Variable
ω_e	Engine Speed
m_f	Fuel Mass per Injection
R_{im}	Gas Constant (Inlet Manifold)
ho_{im}	Specific Stagnation Enthalpy (Inlet Manifold)
γ_{im}	Ratio of Specific Heats (Inlet Manifold)
P_{im}	Gas Pressure (Inlet Manifold)
T_{im}	Gas Temperature (Inlet Manifold)
f_{im}	Gas Fuel/Air Ratio (Inlet Manifold)
m_{im}	Gas Mass (Inlet Manifold)
R_{ex}	Gas Constant (Exhaust Manifold)
ho_{ex}	Specific Stagnation Enthalpy (Exhaust Manifold)
γ_{ex}	Ratio of Specific Heats (Exhaust Manifold)
P_{ex}	Gas Pressure (Exhaust Manifold)
T_{ex}	Gas Temperature (Exhaust Manifold)
f_{ex}	Gas Fuel/Air Ratio (Exhaust Manifold)
m_{ex}	Gas Mass (Exhaust Manifold)
θ_s	Static Timing

Table 7.5a Engine Variables Supplied to the Cylinder Control Volume Tasks during the Corrector Calculations.

Symbol	Name of Variable	
ω_e	Engine Speed	
ω_{tc}	Turbocharger Speed	
P	Gas Pressure	
T	Gas Temperature	for each
f	Gas Fuel/Air Ratio	connected
R	Gas Constant	cylinder
γ	Ratio of Specific Heats	
ho	Specific Stagnation Enthalpy	
A	Effective Flow Area of Valve	

Table 7.5b Engine Variables Supplied to the Manifold Control Volumes, During the Corrector Calculations.

Packet Action Type	Operation
act.position	Return current gas variables
act.displaypos (1)	Return engine variables required by the display task to generate the graphics display (see Table 7.9)
act.cycle (1)	Return control volume steady state performance
act.logstart	Enable data logging
act.logstop	Disable data logging
act.loggasresponse	If data logging is enabled, record control volume gas response
act.selfe	Eliminate control volume task

(1) cylinder control volumes only

Table 7.6 Miscellaneous Operations Performed by the Control Volume Tasks.

Packet Action Type	Operation
act.init	Set up initial shaft conditions
act.position	Return current shaft responses
act.send	Change load system characteristic
act.selfe	Eliminate shaft task

Table 7.7 Miscellaneous Operations Performed by the Shaft Task.

Packet Action Type	Operation
act.init	Set up initial actuator conditions
act.position	Return current actuator responses
act.selfe	Eliminate actuator task

Table 7.8 Miscellaneous Operation Performed by the Actuator Task.

Symbol	Name of Variable
θ	Crankshaft Position
P	Cylinder Gas Pressure
T	Cylinder Gas Temperature
f	Cylinder Gas Fuel/Air Ratio
m	Cylinder Gas Mass
V	Cylinder Gas Volume
ω_e	Engine Speed
m_f	Previous Fuel Mass per Injection
θ_s	Static Timing
θ_d	Dynamic Timing
θ_{id}	Ignition Delay
A_{iv}	Effective Flow Area of Inlet Valve
A_{ev}	Effective Flow Area of Exhaust Valve

Table 7.9 Information Provided to the Display Task, by a Cylinder Control Volume Task.

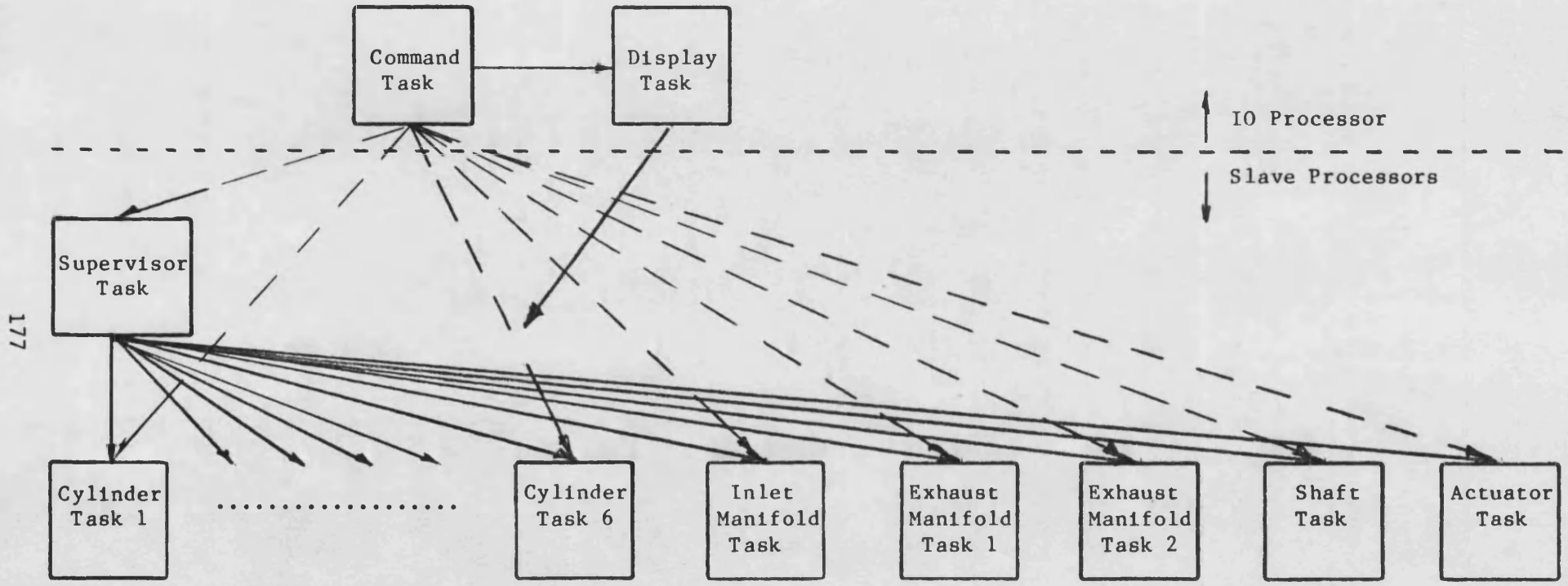


Figure 7.1 Block Diagram Representation of the Engine Simulator Software.

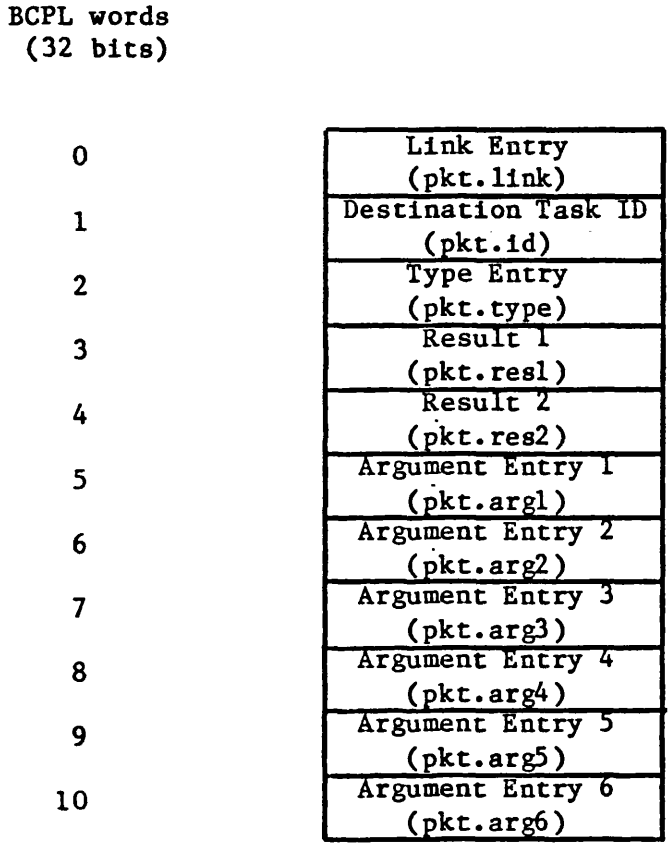


Figure 7.2 Packet Structure.

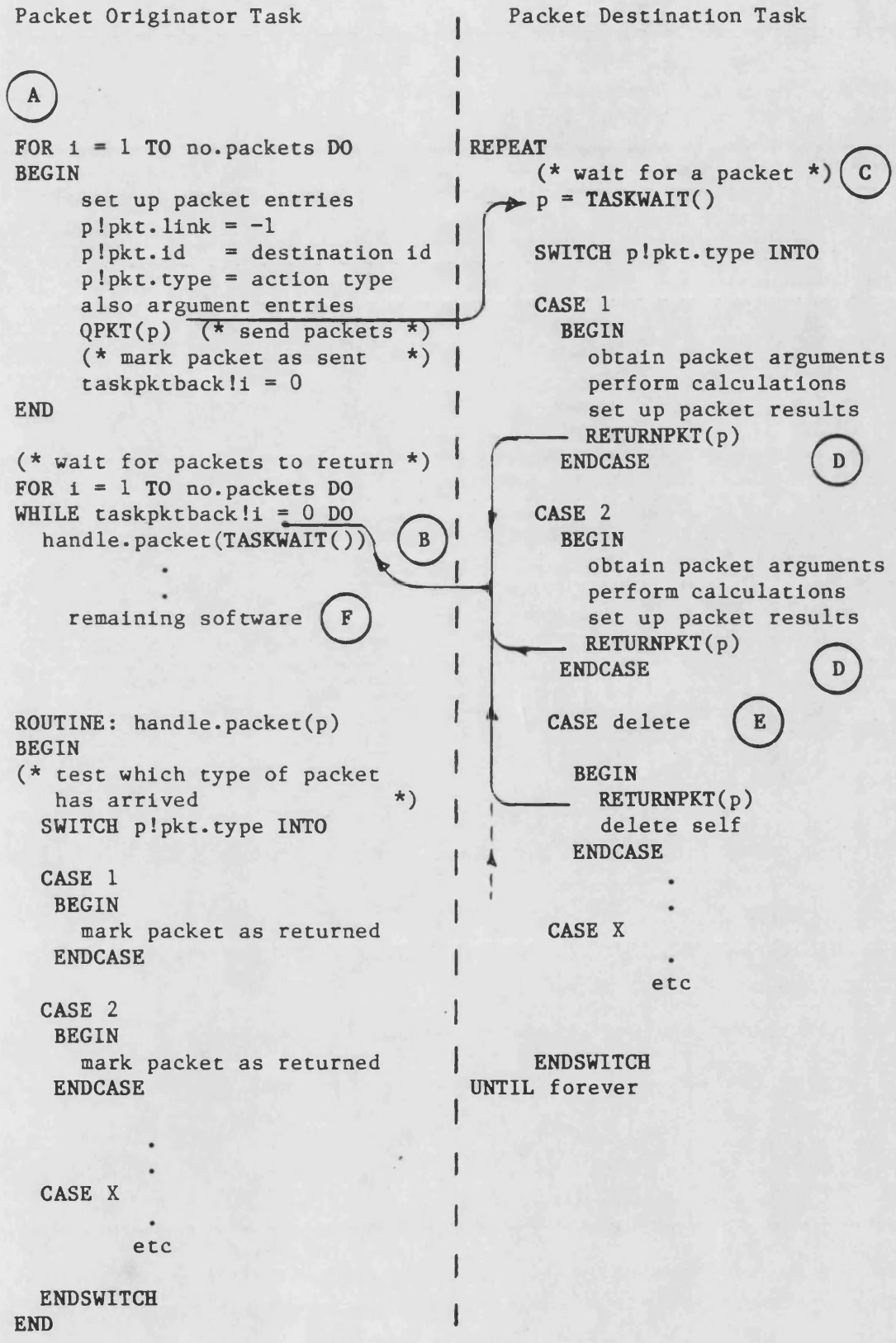


Figure 7.3 Basic Program Code to Manage Packet Transfers between Tasks.

```
MODULE: Command Task

BEGIN

     load simulator tasks to specified processing nodes

          - supervisor task
          - manifold control volume tasks
          - cylinder control volume tasks
          - shaft task
          - actuator task
          - display task

     send simulator tasks a packet specifying initial conditions

     send supervisor task a packet to commence engine model
     calculations

     send display task a packet to commence display of engine model
     performance

     >interactive()

END


SUB-MODULE: interactive()

REPEAT

     display prompt at console

     wait for operator to enter a command

     perform command entered by operator

UNTIL forever
```

Figure 7.4 Outline Pseudo-Code for the Command Task.

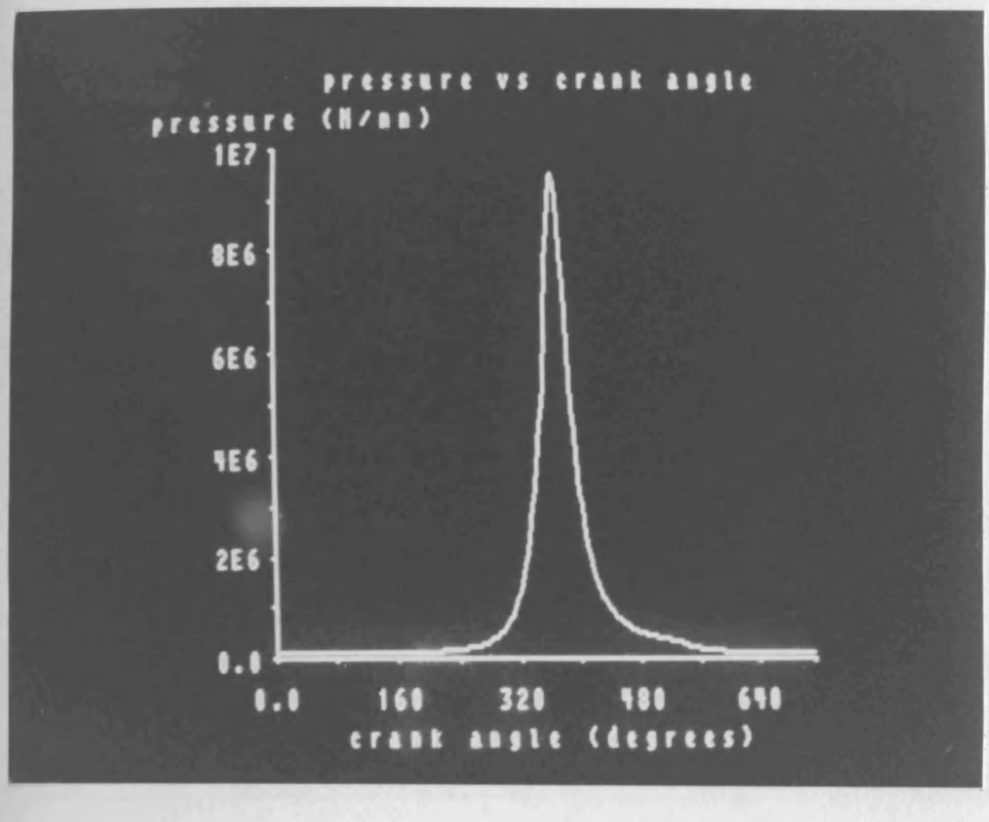


Figure 7.5a Cylinder Gas Pressure vs Crankshaft Angle.

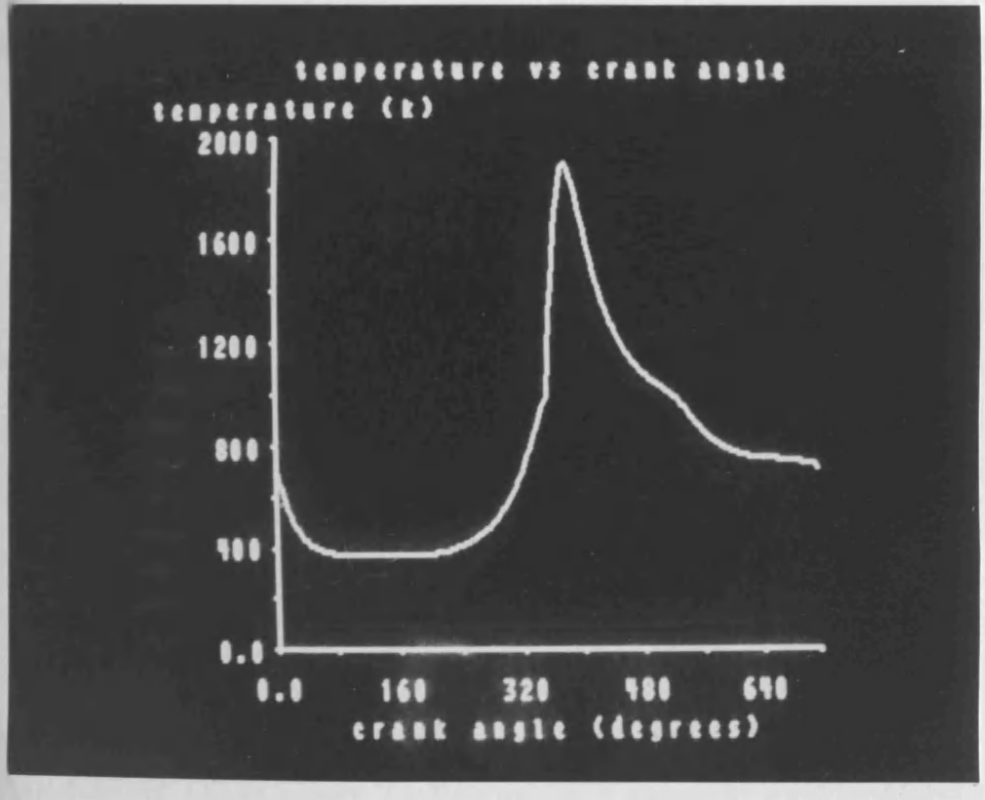


Figure 7.5b Cylinder Gas Temperature vs Crankshaft Angle.

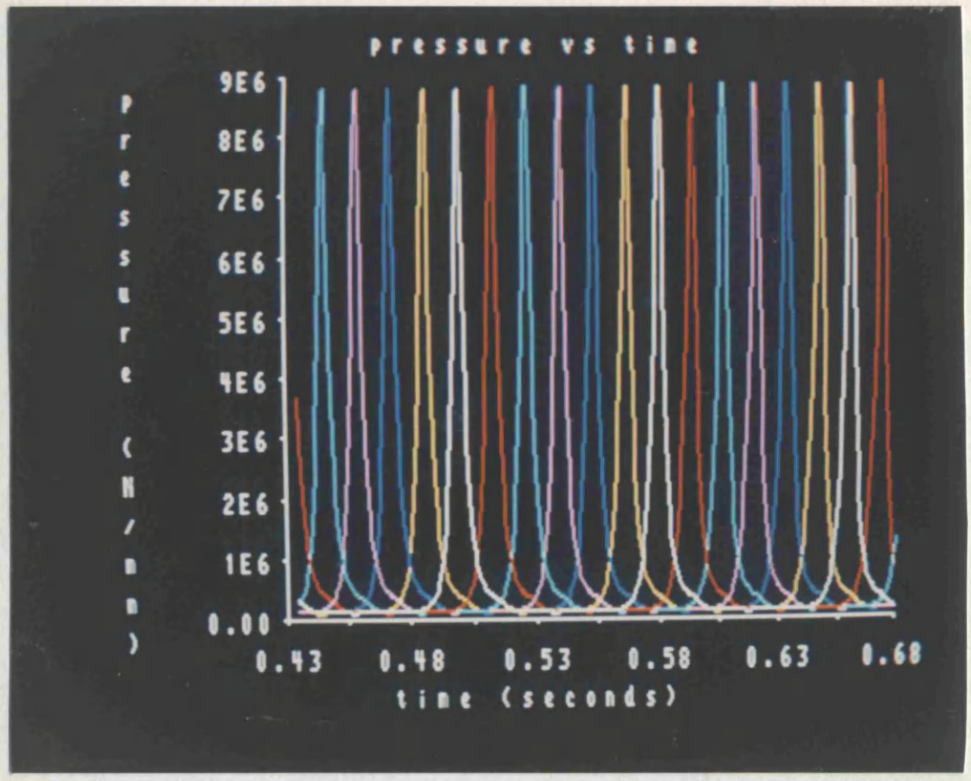


Figure 7.5c Cylinder Gas Pressure vs Time for a Period of Three Power Cycles.

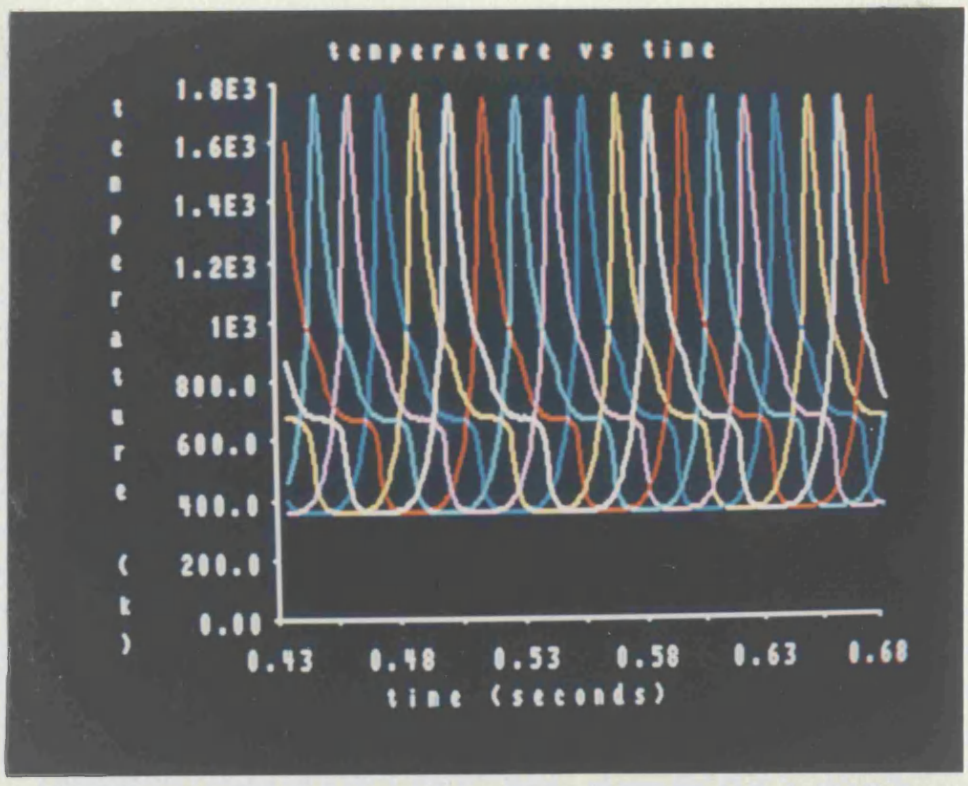


Figure 7.5d Cylinder Gas Temperature vs Time for a Period of Three Power Cycles.

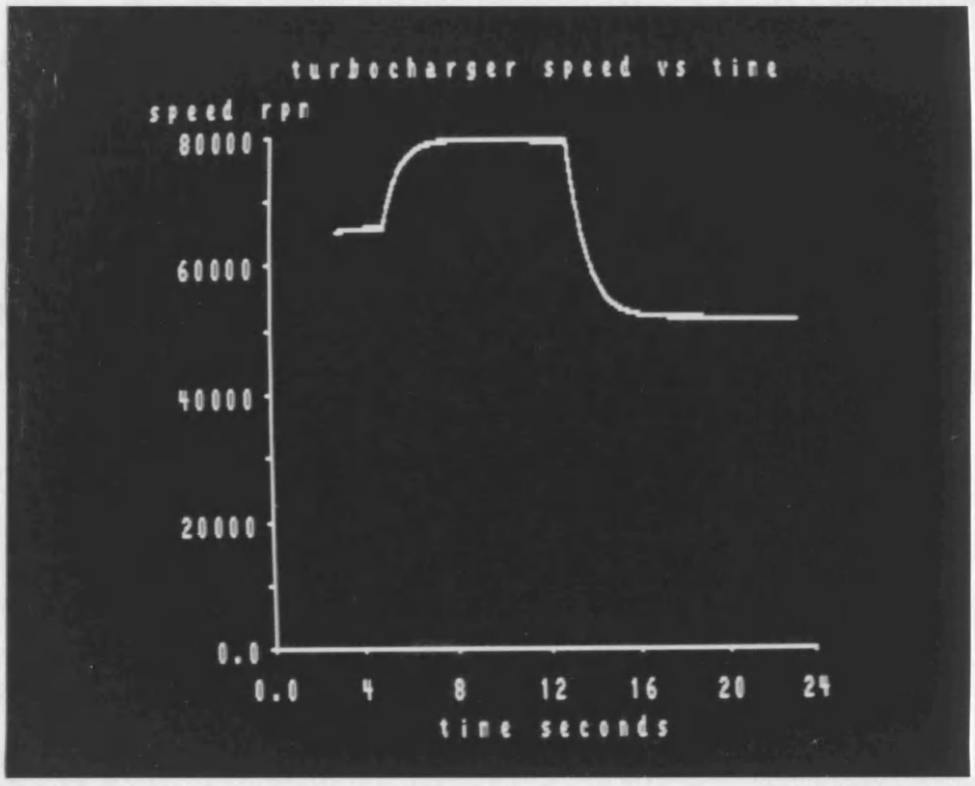


Figure 7.5e Response of Turbocharger Speed to Step Changes in the Position of the Fuel Rack.

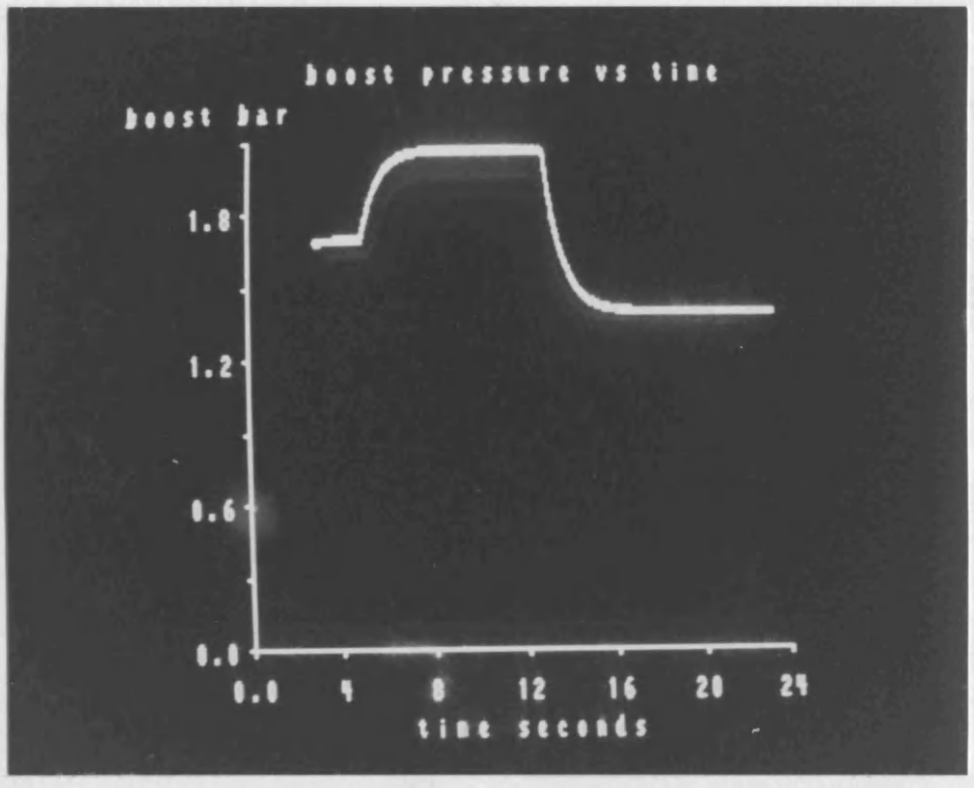


Figure 7.5f Response of Boost Pressure to Step Changes in the Position of the Fuel Rack.

```
MODULE: supervisor task
BEGIN

     perform task initialisation

     REPEAT handle.packet(TASKWAIT)  (* wait for the command task
     UNTIL initilized = TRUE         to send a "act.init" packet *)

     get vectors used by model tasks to return their results in

     send model tasks a packet to calculate their initial conditions
     "act.init"

     set up packet structures for engine model tasks

     > solve.model()

END { module supervisor }


(* this sub-module manages the engine model solution proceedure *)

SUB-MODULE: solve.model
BEGIN
   initialise sub-module variables

   REPEAT
   (* advance crankshaft angle from cal, at which the gas
   |  conditions are known to stopca *)
   |  IF cal = 12.566372                    (* @ 720 degrees ?
   |           THEN BEGIN                      then set crankshaft
   |                    cal        = 0.0       angle to 0 degrees *)
   |                    ca.nextlog = 0.0
   |                    stopca     = stepsize
   |                END
   |           ELSE BEGIN
   |                    stopca     = cal + stepsize
   |                    IF stopca > 12.566372 THEN stopca = 12.566372
   |                END
   |
   |  (* perform data logging *)
   |  IF cal >= ca.nextlog THEN send engine tasks a packet to
   |                             record their responses
   |  stepca     = stepsize   (* set primary integration step size *)
   |
   |  REPEAT  (* ca2 is the crankshaft angle at which an attempt is
   |  |          to be made to solve the engine model              *)
   |  |
   |  |    ca2 = cal + stepca          (* increment crankshaft angle *)
   |  |    IF ca2 > stopca THEN BEGIN
   |  |                            ca2    = stopca
   |  |                            stepca = ca2 - cal
   |  |                         END
```

Figure 7.6 Pseudo-Code for the Supervisor Task (continued p.185)

```
MODULE: supervisor task
BEGIN

     perform task initialisation

     REPEAT handle.packet(TASKWAIT)  (* wait for the command task
     UNTIL initilized = TRUE         to send a "act.init" packet *)

     get vectors used by model tasks to return their results in

     send model tasks a packet to calculate their initial conditions
     "act.init"

     set up packet structures for engine model tasks

     > solve.model()

END { module supervisor }


(* this sub-module manages the engine model solution proceedure *)

SUB-MODULE: solve.model
BEGIN
   initialise sub-module variables

   REPEAT
   (* advance crankshaft angle from cal, at which the gas
     conditions are known to stopca *)
     IF cal = 12.566372                    (* @ 720 degrees ?
             THEN BEGIN                       then set crankshaft
                       cal        = 0.0       angle to 0 degrees *)
                       ca.nextlog = 0.0
                       stopca     = stepsize
                  END
             ELSE BEGIN
                       stopca     = cal + stepsize
                       IF stopca > 12.566372 THEN stopca = 12.566372
                  END

     (* perform data logging *)
     IF cal >= ca.nextlog THEN send engine tasks a packet to
                               record their responses
     stepca     = stepsize   (* set primary integration step size *)

     REPEAT  (* ca2 is the crankshaft angle at which an attempt is
                to be made to solve the engine model                 *)

        ca2 = cal + stepca          (* increment crankshaft angle *)
        IF ca2 > stopca THEN BEGIN
                                 ca2    = stopca
                                 stepca = ca2 - cal
                             END
```

Figure 7.6 Pseudo-Code for the Supervisor Task (continued p.185)

```
    (* apply predictor *)

    send engine model tasks a packet to apply the predictor
    equation to their gas rates at cal, to obtain estimates of
    the engine model states at ca2

    (* apply corrector *)

    REPEAT
        corrector.count = 0     (* initialise corrector count *)
        stability.ok    = TRUE (* initialise stability flag  *)

        REPEAT
            corrector.count = corrector.count + 1

            set up control volume model boundary conditions

            send control volume tasks a packet to evaluate
            their state equations, apply the corrector formula
            and apply the stability critera

        UNTIL (stability.ok = TRUE) OR (corrector.count = 3)

        IF stability.ok
            THEN send engine model tasks a packet to update
                 their state variable values etc
            ELSE (* reduce integration step size by half *)
                 BEGIN
                       stepca = stepca*0.5
                       ca2    = cal + stepca

                       send engine model tasks a packet to
                       apply the predictor equation to their
                       gas rates at cal, to obtain estimates
                       of the engine model states at new value
                       of ca2
                 END

    UNTIL stability.ok

    cal = ca2                   (* update crankshaft angle *)
  UNTIL cal = stopca

UNTIL go = FALSE                (* unless a "self-eliminate packet
                                   has been received continue   *)
  eliminate self
END { of sub-module solve.model }
```

Figure 7.6 Pseudo-Code for the Supervisor Task (continued p.186)

```
(* this sub-module handles newly arrived and returned packets *)


SUB-MODULE: handle.packet(p)
BEGIN
    SWITCHON p!pkt.type INTO      (* switch packet type *)
    |
    |  (* newly arrived packets from the command task *)
    |  CASE act.init :               (* initialisation packet *)
    |
    |    initialise supervisor task
    |     ie obtain information concerning engine configuration and
    |     the primary integration step size to use etc.
    |
    |    set up data structures for communication between
    |    supervisor task and engine model tasks
    |
    |    initilized = TRUE         (* set flag *)
    |    RETURNPKT(p)              (* return initialisation packet *)
    |
    |  ENDCASE
    |
    |  CASE act.selfe :              (* self eliminate packet *)
    |    selfe.pkt = p             (* remember packet pointer *)
    |    go = FALSE                (* set flag eliminate flag *)
    |  ENDCASE
    |
    |  CASE act.send :               (* change control inputs packet *)
    |    obtain new demand control actuator settings from the packet
    |    RETURNPKT(p)               (* return "send" packet *)
    |  ENDCASE
    |
    |  (* packets returned from the engine model tasks *)
    |
    |  CASE act.update:              (* update packets returned *)
    |    mark packet as returned
    |  RETURN
    |
    |  CASE act.loggasresponse:      (* data log packets returned *)
    |    mark data log packet as returned
    |  RETURN
    |
    |  CASE act.predict:             (* predictor packets returned *)
    |    mark predictor packet as returned
    |  RETURN
    |
    |  CASE act.corrector:           (* corrector packets returned *)
    |    mark corrector packet as returned
    |    stability.ok = stability.ok AND stability result returned
    |                                    by the control volume model
    |  RETURN
    |
    ENDSWITCH
END { sub-module handle.packet }
```

Figure 7.6 Pseudo-Code for the Supervisor Task.

```
(* cylinder control volume task *)

MODULE: cylinder task
BEGIN
     perform cylinder task initialization

     REPEAT
       pkt = TASKWAIT()              (* wait for a packet to arrive *)
       SWITCHON pkt!pkt.type INTO          (* find out packet type *)

       CASE act.init:             (* initialization packet received *)

           set up initial conditions for the cylinder eg gas
           pressure temperature etc (see Table 7.3)

           calculate gas properties
           calculate cylinder geometry, eg volume, wall area

           RETURNPKT()          (* return the packet together with the
       ENDCASE           results of the initialisation calculations *)

       CASE act.predict:                (* predictor packet received *)

           from the crankshaft angle evaluate which gas state
           equations are to be used, scavenge, induction etc.

           apply predictor integrator to the gas rates ar θn

           calculate cylinder geometry at θn+1 (volume etc)
           calculate gas properties using predicted state variables
           calculate gas pressure using predicted state variables
           IF cylinder is operating in the combustion phase
           THEN  evaluate combustion model
           RETURNPKT()           (* return the predictor packet with
       ENDCASE                        predictor calculation results *)

       CASE act.corrector:              (* corrector packet received *)

          get boundary conditions to use in calculating gas
          state equations (supplied with packet)

          > calculate.states       (* evaluate gas state equations *)

          apply corrector numerical integrator to the gas rates

          test stability critera and set a stability flag

          calculate gas properties using corrected state variables
          calculate the gas pressure using corrected values

          RETURNPKT()   (* return corrector packet with corrector
       ENDCASE                                    calculation results *)
```

Figure 7.7 Pseudo-Code for Cylinder Task (continued on p.188)

```
CASE act.update:                    (* update packet received *)

   update cylinder state variables, rates of change and
   crankshaft position
   RETURNPKT()                        (* return update packet *)

ENDCASE

CASE act.displaypos :      (* display task packet received *)

   set up engine variables required by the display task
   (see Table 7.9) to generate the graphics display
   RETURNPKT()              (* return packet with variables *)
ENDCASE

CASE act.position :      (* position packet received from
                                              the command task *)
   set up current cylinder variables (pressure, temperature
   etc in the packet
RETURNPKT()                 (* return packet with variables *)
ENDCASE

CASE act.logstart:  (* start data logging packet received *)

   log.start = TRUE           (* set data logging flag true *)
   RETURNPKT()                           (* return the packet *)
ENDCASE

CASE act.logstop:     (* stop data logging packet received *)

   log.start = FALSE         (* set data logging flag false *)
   RETURNPKT()                           (* return the packet *)
ENDCASE

CASE act.rates:            (* evaluate state equation packet
                                                      received *)
   set up boundary conditions (supplied with packet)
   > calculate.states
   RETURNPKT()
ENDCASE

CASE act.selfe:           (* self eliminate packet received *)
   RETURNPKT()
   delete self
ENDCASE

CASE act.loggasresponse:      (* log gas response received *)
   IF data logging enabled THEN log gas responses
   RETURNPKT()
ENDCASE
```

Figure 7.7 Pseudo-Code for Cylinder Task (continued on p.189)

```
      |  CASE act.cycle:      (* steady state data packet received *)
      |     set up steady state engine cycle data in packet
      |     RETURNPKT()
      |  ENDCASE
      |
      |  ENDSWITCH
     UNTIL forever
END  { module cylinder }


(* this module evaluates which set of state equations are to be used
   and then calls the appropriate routine                          *)

SUB-MODULE calculate.states

BEGIN
     SWITCHON crankshaft.position INTO
        CASE flag.scavenge:        > scavenge.routine()
                                     ENDCASE

        CASE flag.induct:          > induction.routine()
                                     ENDCASE

        CASE flag.comp:            > compression.routine()
                                     ENDCASE

        CASE flag.comb:            > combustion.routine()
                                     ENDCASE

        CASE flag.power:           > powerstroke.routine()
                                     ENDCASE

        CASE flag.exh:             > exhaust.routine()
                                     ENDCASE

     ENDSWITCH
END {module calculate.states}


(* typical example of a state equation routine - this one for the
   scavenge phase of engine operation                              *)

SUB-MODULE: scavenge.routine
BEGIN
     calculate heat transfer between cylinder gas and walls
     calculate flow of gas through the inlet and exhaust valves

     calculate rate of change of gas mass  dm/dθ
     calculate rate of change of gas fuel/air ratio df/dθ
     calculate rate of change of gas temperature  dT/dθ

END  {module scavenge.routine }
```

Figure 7.7 Pseudo-Code for Cylinder Task.

```
MODULE: shaft task
BEGIN
     initialise shaft task

     REPEAT
     |    pkt = TASKWAIT()          (* wait for a packet to arrive *)
     |
     |    SWITCHON pkt!pkt.type INTO       (* find out packet type *)
     |
     |    CASE act.calc :          (* calculate shaft speeds packet *)
     |
     |          obtain engine brake torque, turbine torque,
     |          compressor torque and the integration step size from
     |          the packet
     |
     |          calculate engine load torque
     |
     |          calculate angular acceleration of the engine and
     |          turbocharger shaft
     |
     |          integrate acceleration to obtain shaft speeds
     |
     |          RETURNPKT()     (* return packet with shaft speeds *)
     |     ENDCASE
     |
     |     CASE act.update:               (* update packet received *)
     |          update shaft speeds
     |          RETURNPKT()                 (* return update packet *)
     |     ENDCASE
     |
     |     CASE act.init :       (* initialisation packet received *)
     |          set up initial shaft conditions (eg load system
     |          viscous damping etc )
     |          RETURNPKT()
     |     ENDCASE
     |
     |     CASE act.position :                    (* position packet *)
     |          return shaft responses (speeds, acceleration etc)
     |          RETURNPKT()
     |     ENDCASE
     |
     |     CASE act.send    :                 (* change load system *)
     |          change characteristics of the engine load system
     |          RETURNPKT()
     |     ENDCASE
     |
     |     CASE act.selfe :      (* self eliminate packet received *)
     |          RETURNPKT()
     |          delete self
     |     ENDCASE
     |     ENDSWITCH
     UNTIL forever
END
```

Figure 7.8 Pseudo-Code for Shaft Task.

```
MODULE: actuator task
BEGIN
     initialise actuator task

     REPEAT
       pkt = TASKWAIT()          (* wait for a packet to arrive *)

       SWITCHON pkt!pkt.type INTO

       CASE act.calc :      (* calculate engine control settings *)

         get demanded actuator positions, engine speed and the
         integration step size from the packet

         calculate the acceleration of the actuators
         integrate actuator accelerations to obtain velocity
         apply slew rate limit to actuator velocity
         integrate actuator velocity to obtain actuator position
         apply end stop limits to actuator position

         using engine speed and fuel rack actuator position
         evaluate fuel delivery
         evaluate static fuel injection timing
         evaluate turbine restriction

         RETURNPKT()   (* return the packet together with the
                                  results of the calculations *)
       ENDCASE

       CASE act.update:                (* update packet received *)
         update actuator state variables
         crankshaft position
         RETURNPKT()                     (* return update packet *)
       ENDCASE

       CASE act.init :         (* initialisation packet received *)
         set up initial actuator positions etc
         RETURNPKT()
       ENDCASE

       CASE act.position :                    (* position packet *)
         return actuator responses (positions etc)
         RETURNPKT()
       ENDCASE

       CASE act.selfe :        (* self eliminate packet received *)
         RETURNPKT()
         delete self
       ENDCASE
       ENDSWITCH
     UNTIL forever
END
```

Figure 7.9 Pseudo-Code for Actuator Task.

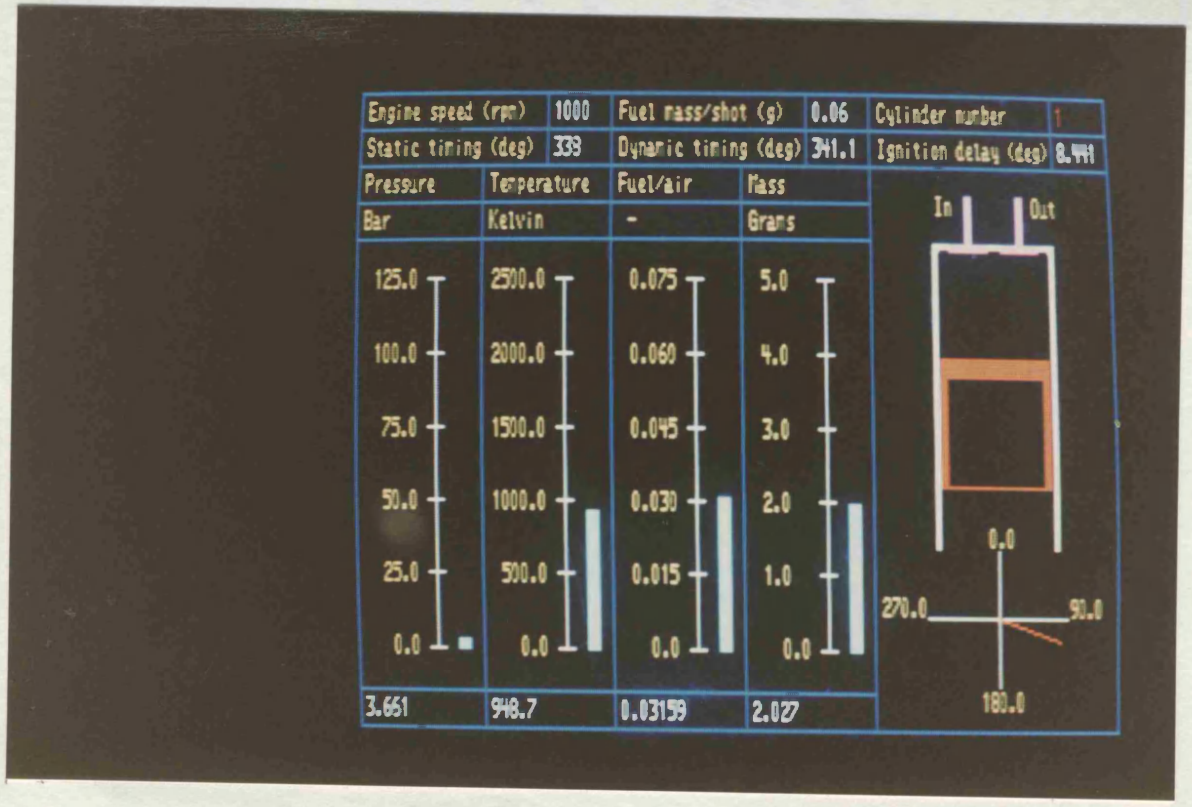


Figure 7.10 Cylinder Control Volume Display.

```
MODULE: display task
BEGIN

     initialise display task
     RETURNPKT()                          (* start display packet *)

     draw static picture elements on both graphics pages

     REPEAT
          REPEAT

               send cylinder control volume model a "displaypos"
               packet to return data required to generate display

               undraw old moving picture elements
               draw new moving picture elements

               swap graphics page

          UNTIL  stopped = TRUE

          REPEAT handle.packet(TASKWAIT())
          UNTIL display.on packet received
     UNTIL forever
END { module display task }


SUB-MODULE: handle.packet(p)
BEGIN
     SWITCHON p!pkt.type INTO         (* find out the packet type *)

          CASE act.dispoff :       (* halt display packet received *)
             stopped = TRUE                              (* set flag *)
             RETURNPKT()
          ENDCASE

          CASE act.dispon :     (* restart display packet received *)
             stopped = FALSE                             (* set flag *)
             RETURNPKT()
          ENDCASE

          CASE act.selfe :                (* self eliminate packet *)
             mark self eliminate packet as received
          ENDCASE

          CASE act.display :         (* change cylinder for display *)
             obtain task id of cylinder for display from the packet
             RETURNPKT()
          ENDCASE

     ENDSWITCH
END { module handle.packet }
```

Figure 7.11 Pseudo-Code for Display Task

CHAPTER 8

8.1 Software Testing

The ability of filling and emptying models to accurately predict the steady state and dynamic performance of diesel engines has been adequately demonstrated by others [8.1-8.3], and there would be little purpose in seeking to re-establish this. Even so, it was considered essential to confirm that the engine model used in this research can produce results which are similar to the performance of the real TL11 engine. Confirmation of this was seen as an important step because it demonstrates that the model had been correctly implemented and that no coding errors of any significance remain in the model software. Had such errors remained, they may have influenced the model run times and perhaps could throw doubt on the validity of the results showing how much faster the engine model is executed when computed concurrently (see Chapter 9).

Great care was taken during development of the simulator software to eliminate coding errors at an early stage. Each simulator task was tested as a separate entity using known initial conditions and the results compared with hand calculated results: this activity involved the development of several test programs.

Having ensured that each of the simulator tasks performed as expected, attention was then focused on the complete simulator. First, the simulator results were inspected to confirm that they were "sensible" and then the simulator was used to calculate steady state performance results for experiments which had been performed

on the real engine, and the results were compared. Finally, the dynamic response of the engine model was obtained to step disturbance signals and the results compared with those obtained during similar experiments on the real engine. It will be shown that the results obtained using the engine simulator are in adequate agreement with those obtained using the real engine; the results are described below in some detail, not only to verify that the model has been correctly implemented, but also to demonstrate the capabilities of the engine simulator.

The software testing is presented in three parts. Section 8.2, presents results which show the behaviour of the gas in the engine cylinders and manifolds, Section 8.3, the steady state performance of the engine when it is operating on its limiting torque curve, and finally Section 8.4, the dynamic performance of the engine when subjected to various changes to the control inputs.

8.2 Engine Gas Behaviour

This section presents results which show how the gas in the engine cylinders and manifolds behaves. Two sets of results are presented, the first (Section 8.2.1) illustrates the behaviour of the gas when the engine is operating under steady conditions, and the second (Section 8.2.2), the effect of a change in the engine control inputs on the gas conditions in the engine cylinders.

8.2.1 Gas Behaviour under Steady Operating Conditions

The results presented below were obtained using the TL11 engine model operating at a steady speed of 1500 rpm and supplied with 0.09 grams of fuel every injection; fuel injection timing was set to 22 degrees btdc and the turbine nozzle control was set to its fully open position. Once the model solution had reached a steady condition, the behaviour of the gas in the engine manifolds and cylinders was recorded, and these records are shown in Figures 8.1 to 8.17. With the exception of the pressure-volume diagram (Figure 8.10), all the responses are plotted against crankshaft position, over a complete engine cycle, which at the simulated engine speed of 1500 rpm, would be completed by the real engine in 80 ms.

8.2.1.1 The Response of the Gas in a Cylinder

When the engine is running at a steady operating condition, the gases in the six engine cylinders behave identically to one another, except that the responses are, of course, phased. Figures 8.1 to 8.10 show how the gas in one cylinder behaves during the power cycle.

Figure	
8.1	Rate of change of gas temperature
8.2	Gas temperature
8.3	Rate of change of gas fuel-air ratio
8.4	Gas fuel-air ratio

8.5 Rate of change of gas mass

8.6 Mass of gas in the cylinder

8.7 Gas pressure

8.8 Rate of heat transfer

8.9 Work

8.10 Pressure-volume diagram for the cycle

These results will now be discussed in some detail.

- **The period from the inlet valve opening to the inlet valve closing (710 - 230 degrees).**

  This period covers the scavenge and induction phases of engine operation in which a fresh charge of air is drawn into the cylinder. Figure 8.1 shows that during the early stages of this phase, the rate of change of gas temperature is negative as "cool" air is drawn into the cylinder from the inlet manifold. This lowers the temperature of the gas remaining in the cylinder from the previous power cycle (Figure 8.2). The temperature of the gas continues to fall until it is approximately equal to the temperature of the gas in the inlet manifold, and then remains at about this temperature until the inlet valve closes.

  During the early stages of this phase the rate of change of fuel-air ratio is also negative (Figure 8.3) and consequently, the gas fuel-air ratio reduces (Figure 8.4). This reduction is also due to the flow of fresh air from the inlet manifold mixing with the residual gas in the cylinder.

  The rate of change of mass of gas in the cylinder is shown in

Figure 8.5. The exhaust valve closes at a crankshaft position of 14 degrees, and from then until the inlet valve closes (230 degrees), the rate of change of mass is due entirely to flow of gas through the inlet valve. As the piston moves down the cylinder, the gas flows from the inlet manifold into the cylinder, and the mass of gas in the cylinder increases, as shown in Figure 8.6. The piston reverses its direction of motion at a crankshaft angle of 180 degrees, subsequently reducing the volume of gas in the cylinder. However, the inlet valve does not close until a crankshaft angle of 230 degrees and consequently, during the early stages of the compression stroke, some gas flows back from the cylinder into the inlet manifold (as Figure 8.6 shows). During this period the rate of change of mass is negative (Figure 8.5).

- **The period from the inlet valve closing to fuel ignition (230 - θ_i degrees).**

  Once the inlet valve has closed, no more mass enters or leaves the cylinder until the injection of fuel takes place. Consequently, during this period, the rate of change of fuel-air ratio, and the rate of change of mass in the cylinder are both zero, as is shown in Figures 8.3 and 8.5, respectively. The gas trapped in the cylinder when the inlet valve is closed, is compressed as the piston moves up the cylinder, and as a result the rate of change of temperature is positive, as shown in Figure 8.1. Consequently, the pressure and temperature of the gas in the cylinder increase (Figure 8.2 and Figure 8.7). The rate of heat transfer between the

cylinder gas and the combustion chamber walls increases markedly (Figure 8.8) and the work which the piston performs in compressing the gas, is shown in Figure 8.9.

- **The period from fuel injection, to the exhaust valve opening (θ_i - 494 degrees).**

  In this period, fuel is injected into the cylinder, ignition occurs, and the fuel burns releasing heat energy. The instant of ignition shows up clearly as a discontinuity in the rate of change of temperature, rate of change of fuel-air ratio, and rate of change of mass in the cylinder, as is seen from Figures 8.1, 8.3 and 8.5 respectively. Figure 8.5 shows that the early phase of rapid pre-mixed burning of the fuel (see Chapter 4) lasts for only a few degrees of crankshaft rotation, after which the slower diffusion mode of burning takes over. The high rate of heat release from the burning fuel causes the pressure and temperature of the gas in the cylinder to rise sharply, as shown in Figure 8.2 and Figure 8.7. The rate of heat transfer between the cylinder gas and combustion chamber wall also increases rapidly during the early period of combustion, as shown in Figure 8.8. Once the piston has travelled beyond TDC, the gas drives the piston back down the cylinder and makes a large positive work contribution (Figure 8.9). When the combined effects of work and heat transfer exceed the heat energy released by the burning fuel, the rate of change of gas temperature becomes negative (Figure 8.1) and the gas temperature then falls rapidly (Figure 8.2).

The fuel-air ratio of the gas in the cylinder increases rapidly as the fuel burns (Figure 8.4), and the mass of gas in the cylinder increases by an amount equal to the mass of fuel burnt (Figure 8.6). Once combustion has finished, the rates of change of fuel-air ratio (Figure 8.3) and mass (Figure 8.5) become zero, since no mass is then able to enter, or leave, the cylinder until the exhaust valve opens.

- **Period from the exhaust valve opening, to the inlet valve opening (494 - 710 degrees).**

  During this period, the exhaust valve opens and the hot exhaust gas in the cylinder is expelled into the exhaust manifold. The instant at which the valve opens is clearly seen from the rate of change of mass which goes sharply negative as gas starts to flow from the cylinder (Figure 8.5); consequently the mass of gas in the cylinder reduces as shown in Figure 8.6. Figure 8.5 shows that the rate of mass flow through the exhaust valve increases rapidly as the valve opens. As the gas flows out of the cylinder into the exhaust manifold, the gas pressure in the cylinder gradually falls (Figure 8.7), and the gas pressure in the exhaust manifold increases. Therefore, the mass flow rate reaches a maximum and then falls as shown in Figure 8.5. Initially, the decline is rapid, but is then checked for a period by the reduction in the cylinder gas volume, as the piston moves up the cylinder. Once the rate of change of cylinder gas volume has passed its peak, the mass flow rate again declines rapidly.

Throughout this period the direction of gas flow is from the cylinder into the exhaust manifold. Hence, the rate of change of fuel-air ratio is zero (Figure 8.3), and the fuel-air ratio of the gas in the cylinder remains unchanged (Figure 8.4). Finally, Figure 8.10 shows the pressure volume diagram for the engine cycle.

8.2.1.2 The Response of the Gas in the Inlet Manifold

Figures 8.11 to 8.13 respectively show how the pressure, temperature and mass of the gas in the inlet manifold vary during the power cycle. The figures show that as each cylinder in turn, draws air from the inlet manifold, there is a slight reduction in the pressure, temperature and mass of gas in the inlet manifold. These variations are quite small because the volume of the inlet manifold is large relative to the volume of the cylinders.

8.2.1.3 The Response of the Gas in the Exhaust Manifold

Figures 8.14 to 8.17 respectively, show how the pressure, temperature, fuel-air ratio and mass of gas in an engine exhaust manifold, vary during the power cycle. The exhaust manifold is supplied with gas from three cylinders and the figures show that the manifold pressure (Figure 8.14), temperature (Figure 8.15) and mass (Figure 8.17) vary considerably as each cylinder, in turn, ejects its hot exhaust gas. The variations are large since the volume of the exhaust manifold is less than the swept volume of an engine cylinder. Figure 8.16 shows that the fuel-air ratio of the manifold

gas is relatively constant having approximately the same value as the fuel-air ratio of the gas ejected from the cylinders, (Figure 8.4). This is to be expected since the fuel-air ratio of the gas flowing from each cylinder into the exhaust manifold is constant, except during the brief scavenge period when the inlet valve opens, and allows fresh air into the cylinder, - thereby reducing the fuel-air ratio of the cylinder gas.

8.2.2 Response of the Gas in the Engine Cylinders to the Engine Control Inputs

The results given in this section illustrate the effect of a step change in the position of the fuel rack actuator, the fuel timing actuator and the turbine nozzle actuator on the gas in the engine cylinders. All the results were obtained with the engine operating at a steady speed of 1500 rpm.

The test procedure was to allow the engine model to reach steady conditions, then apply the test signal. In these tests, the transient response of the gas is of interest and it was therefore necessary to record the gas response in each of the six engine cylinders. The responses were recorded for a period of three power cycles, which at a speed of 1500 rpm, takes the real engine 240 ms to complete.

- **Response of the cylinder gas, to a step change in fuel rack position.**

  During this experiment, the engine was subjected to a step change in demanded fuel rack position from 6.5 mm to 13 mm.

The experiment was conducted with the turbine restriction fully open and the fuel injection timing set to 22 degrees btdc. Figures 8.18 to 8.23 show how the fuel rack actuator, cylinder gas pressure, temperature, fuel-air ratio, mass and indicated torque respectively, respond to the test signal.

Figure 8.18 shows that the response of the fuel rack actuator is slew rate limited, and overshoots the demanded rack position before settling to a steady value. As the rack position increases, so the quantity of fuel injected into the engine cylinders increases, as is seen from the mass response, of Figure 8.22. The increased level of fueling, causes the gas pressure, temperature and fuel-air ratio to rise generally throughout the engine cycle, as is shown in Figures 8.19 to 8.21 respectively. The contribution of the engine cylinders, to (positive) engine torque production, also increases, as shown in Figure 8.23.

o **Response of the cylinder gas to a step change in fuel injection timing.**

During this test the engine was subjected to a step change in demanded fuel injection timing, from 17 to 37 degrees btdc. The turbine restriction was fully open during the experiment and the fuel rack position was 10 mm. Figures 8.24 to 8.29 show how the fuel timing actuator, cylinder gas pressure, temperature, fuel-air ratio, mass and indicated torque respectively, respond to the test signal.

The response of the fuel timing actuator is slew rate limited,

as is shown in Figure 8.24. As the actuator moves to its demanded position, the instant at which fuel is injected into the engine cylinders advances; consequently the fuel ignites earlier, causing the peak gas pressure and temperature to increase as shown in Figures 8.25 and 8.26. In fact, with the fuel injection timing fully advanced, the gas pressure increases so rapidly during the last stages of the compression stroke that it causes a spike in the indicated torque response, as shown in Figure 8.29. Finally, Figures 8.27 and 8.28 respectively, show the effect of the change in fuel injection timing on the fuel-air ratio and mass of gas in the engine cylinders. As is to be expected over such a small time interval, the change in fuel injection timing causes only minor variations in the fuel-air ratio and mass of gas in the cylinders.

- **Response of the cylinder gas to a step change in turbine restriction.**

  During this test the engine was subjected to a step change in demanded turbine restriction, from 5% to 45%. The experiment was conducted with the fuel rack actuator set to 10 mm and fuel injection timing 22 degrees btdc. Figures 8.30 to 8.35 show how the turbine nozzle restriction actuator, cylinder gas pressure, temperature, fuel-air ratio, mass and indicated torque respectively, respond to the test signal.

  Unlike the fuel rack and fuel timing inputs, the turbine restriction input is unable to directly influence the engine combustion process. However, it does affect the combustion

process indirectly by its influence on the gas conditions in the exhaust manifolds, and via the turbocharger, in the inlet manifold. Since the turbocharger has a time constant of the order of one second, these are long term responses and only minor variations were apparent in the cylinder gas response. For example, Figure 8.33 shows that following the step change in turbine nozzle restriction, the maximum value of fuel-air ratio gradually declined. This is because restricting the turbine, in turn increases the speed of the turbocharger, the pressure in the inlet manifold, and thus the mass of air drawn into the cylinders during the induction stroke (as is shown in Figure 8.34). However, the quantity of fuel supplied to the engine remains unchanged, - hence the gas fuel-air ratio declines.

8.3 Steady State Engine Performance

This section presents steady state engine performance results obtained using the engine simulator. To facilitate a comparison with the performance of the real engine, it was decided to use the simulator to repeat the power curve experiments carried out by Roberts [8.4] on the real engine. Roberts performed a series of four experiments, during which he measured the steady state performance of the engine when it was producing its limiting torque. A different turbine nozzle restriction was used in each series of experiments, covering the total operational range of the variable geometry turbine; these were 0%, (corresponding to a fully open turbine nozzle), 25%, 40% and 50%, (50% corresponding to a fully

restricted turbine nozzle).

The predicted engine model performance is given in Tables 8.1a to 8.4a, for 0%, 25%, 40% and 50% turbine nozzle restriction, respectively. For comparison the performance of the real engine is given in Tables 8.1b to 8.4b. The results are plotted as follows:-

Figure

8.36 Brake mean effective pressure (bmep)

8.37 Specific fuel consumption (sfc)

8.38 Engine brake power

8.39 Boost pressure

8.40 Turbocharger speed

8.41 Engine air mass flow

8.42 The compressor map

8.43 Inlet manifold gas temperature

8.44 Mean exhaust manifold gas pressure

8.45 Mean turbine inlet temperature

Figure 8.36 shows that the predicted value of brake mean effective pressure (bmep), is less accurate at the higher values of turbine nozzle restriction. A similar reduction in accuracy is also apparent in specific fuel consumption (Figure 8.37) and engine brake power (Figure 8.38) since these two quantities depend upon bmep.

The response of boost pressure, turbocharger speed and engine air mass flow rate are shown in Figures 8.39 to 8.41 respectively. These quantities have been plotted on the steady state performance map of the compressor (Figure 8.42), to show the compressor

operation for the range of turbine restrictions tested. The compressor operating points obtained at the same engine speed, have been joined together on the figure, and this clearly shows that the engine model overestimates air mass flow through the engine (also shown in Figure 8.41), which seems to be typical of filling and emptying models. The accuracy of these predicted results also reduce with increasing turbine restriction, particularly at low engine speed. The reduction in accuracy is probably due to using a swallowing curve in the turbine model, which was obtained with the turbine fully open (0% restricted) and which will become increasingly less accurate, as the turbine moves away from fully open operation.

Finally, Figures 8.44 and 8.45, show the mean value of exhaust manifold gas pressure and turbine inlet temperature respectively, obtained by low pass filtering their respective gas responses. The predicted gas pressure agrees reasonably well with the measured value; however the predicted turbine inlet temperature is consistently lower than the measured value by as much as 150 K. Although the engine model overestimates the mass flow of air through the engine, it cannot account for all of the temperature difference. Undoubtedly some of the temperature difference results from the energy balance in the engine cylinders being wrong, with too much of the energy supplied by the fuel going to the coolant (heat transfer) and consequently insufficient energy flowing from the cylinders, through the exhaust valves, into the exhaust manifolds.

From the results presented above, it is evident that the predicted steady state performance of the engine exhibits the same

trends as the real engine although the accuracy obtained is not quite so good as has been demonstrated by others. Nevertheless, the predicted performance is generally within 10% of that of the real engine and undoubtedly could be somewhat improved by careful "tuning" of the many semi-empirical factors incorporated in the various sub-models.

8.4 Dynamic Performance Results

Two sets of results are presented in this section showing how the engine responds dynamically to step changes in the engine control inputs. The first set of results, presented in Section 8.4.1, were obtained using the engine model, and the second set of results (Section 8.4.2) show the response of the real engine; finally in Section 8.4.3, a comparison is made between the two sets of responses.

8.4.1 The Dynamic Response of the Engine Model

This section presents results which show how the engine model responds dynamically to step changes in the position of the fuel rack and turbine nozzle controls. All the simulations start with the engine model running at a steady speed of 1500 rpm, with the fuel rack position set to 11 mm (nominal torque 680 Nm), fuel injection timing set to 27 degrees btdc, and turbine nozzle restriction set to 25%. Once the engine responses had achieved steady values, data logging commenced and then, after a short delay, the appropriate test signal was applied. Each simulation lasted for

22 seconds of real time, which at the simulated engine speed of 1500 rpm, equals 550 revolutions of the engine. During the simulations the engine responses were logged every 50 degrees of crankshaft rotation, which corresponds to every 5.5 ms.

The dynamic response of an engine is, of course, very considerably influenced by the load system which it is driving, since it is the combined dynamics of the engine and load which determines the speed response of the engine, - which, in turn, influences other engine responses. This can cause problems when carrying out experiments to measure the dynamic response of an engine, since the engine cannot be tested without the load system attached, and often the dynamic characteristics of the load are not accurately known. In addition, in order to test the engine at different torque/speed operating conditions, it is usually necessary to change the characteristics of the load system, and the results obtained (at the different operating conditions) are not then directly comparable. These difficulties were minimised by the use of a large inertia as the engine load, so that engine speed can be considered to be constant for the short duration of each experiment, and the engine can therefore be treated as a torque producing system.

The response of the engine to the fuel rack and turbine nozzle restriction test signals are now presented.

8.4.1.1 Response of the Engine to the Fuel Rack Disturbance Signal

Two test signals were used to disturb the engine fuel rack and these are shown in Figures 8.46a and 8.46b respectively. The first test signal (Figure 8.46a), created a step change in the fuel rack position from 11 mm to 12 mm at 5 seconds, and then reduced it to 10 mm at 13 seconds. The second signal (Figure 8.46b) step changed fuel rack position from 11 mm to 13 mm at 5 seconds and then reduced it to 9 mm at 13 seconds. Figures 8.47 to 8.54, show how the fuel rack actuator, engine torque, pressure, temperature and fuel-air ratio of the gas in the exhaust manifold, turbocharger speed, pressure and temperature of the gas in the inlet manifold respond to the test signals.

Engine torque (Figure 8.48), exhaust manifold gas pressure (Figure 8.49) and temperature (Figure 8.50), vary significantly from one cycle to another during engine operation. Consequently, to show up their basic underlying trend, these responses were filtered to remove their high frequency components, and the filtered responses are shown in Figures 8.48, 8.49, and 8.50.

The fuel rack actuator tracks the demanded rack position signals very closely, as is shown in Figure 8.47. The resulting change in quantity of fuel supplied to the engine gives rise to the variation in engine torque shown in Figure 8.48. The change in fueling also causes the pressure, temperature and fuel-air ratio of the gas in the exhaust manifolds to change very significantly, as is shown in Figures 8.49 to 8.51 respectively. Not surprisingly, these changes in exhaust manifold conditions significantly affect

turbocharger speed, (as is shown in Figure 8.52). From Figure 8.42 (which shows the steady state performance map of the turbocharger compressor), it will be seen that the change in turbocharger speed, which was of the order of several thousand rpm, must in turn, result in significant changes to the gas conditions in the inlet manifold and this is confirmed by results shown in Figures 8.53 (gas pressure) and 8.54 (gas temperature). The mass flow of air through the engine increases with boost pressure, and is the principal cause of the gradual reduction in the temperature of the gas in the exhaust manifold (Figure 8.50) and its fuel-air ratio (Figure 8.51), following the step increase in fueling at 5 seconds. A similar, but opposite effect, also occurs when fueling is reduced at 13 seconds.

8.4.1.2 Response of the Engine to the Turbine Nozzle Restriction Disturbance Signal

Two step test signals, were used to disturb the turbine nozzle restriction input, and these are shown in Figure 8.55a and 8.55b. The first test signal, (Figure 8.55a) increased the turbine restriction from 25% to 35% at 5 seconds and then reduced it to 15% at 13 seconds. The second test signal (shown in Figure 8.55b) increased the turbine restriction from 25% to 45% at 5 seconds and then reduced it to 5% at 13 seconds. Figures 8.56 to 8.63 show how the turbine nozzle restriction actuator, engine torque, pressure and temperature of the gas in the exhaust manifold, turbocharger speed, pressure and temperature of the gas in the inlet manifold, respond to the test signals.

The ease with which exhaust gas can flow from the exhaust manifolds depends upon the flow restriction downstream of the manifolds, and this, of course, includes the turbine restriction. Consequently when the turbine restriction is changed, the pressure and temperature of the gas in the exhaust manifolds change (as shown in Figure 8.59 and 8.60). In turn the turbocharger speed changes (Figure 8.61), which causes the pressure and temperature of the gas in the inlet manifold to change, as shown in Figures 8.62 and 8.63 respectively.

The increase in exhaust manifold gas pressure (Figure 8.59), following the step change in turbine restriction at 5 seconds, increases the work that the engine has to perform to expel the hot exhaust gases from the cylinders into the exhaust manifolds. Consequently, the useful torque developed by the engine decreases, as shown in Figure 8.57 for the 20% amplitude test signal, and Figure 8.58 for the 10% amplitude test signal. The reduction in engine torque occurs even though the increase in turbine nozzle restriction improves the engines air supply. This confirms, not surprisingly, that at the engine operating condition simulated, the engine was already receiving sufficient air to burn all the fuel supplied, and that adding to the air supply merely increased the excess over that required for good combustion.

8.4.2 The Dynamic Response of the TL11 Engine

A part of the early investigations carried out during this programme of work, was to obtain black-box models of the TL11 engine responses [8.5] using step and pseudo random binary sequence (prbs)

test signals. The time responses of the engine to the step disturbance signals are useful, since they can be compared with the responses obtained using the filling and emptying engine model.

8.4.2.1 Description of the Test Equipment

A digital computer was used to generate the test signal and to record the response of the engine; details of the computer system are given in Table 8.5. A signal conversion card was designed specially for the research programme, containing four 12 bit digital to analogue converters and 8 multiplexed 12 bit analogue to digital converter channels, each having a conversion time of 35 micro seconds. A schematic of the signal conversion card is shown in Figure 8.64.

The software to generate the test signals and record the engine responses was written using the BCPL computer program language. The special low level routines, which are required to service the analogue to digital converter, digital to analogue converter and timer, were written using an assembler.

8.4.2.2 Test Procedure

The test procedure was to drive the engine to a speed of 1500 rpm and load torque of 500 Nm (rack 9.1 mm); after the engine had reached its normal operating temperature the test signal was applied. The response of the engine to the fuel rack disturbance signal, was measured with the turbine nozzle restriction input held constant at 25%. The response of the engine to the turbine nozzle

disturbance signal, was then measured with the fuel rack position maintained constant. During both experiments fuel injection timing was set to 22 degrees btdc.

During application of a test signal, the computer recorded the engine response at intervals of 40ms, which at an engine speed of 1500 rpm corresponds to one record being taken for each engine revolution. The quantities recorded were: the position of the fuel rack and turbine nozzle actuators, engine and turbocharger rotational speed, load torque, and the pressure of the gas in the inlet and exhaust manifolds.

The dynamometer was operated in its "constant" speed mode, in which it attempts to simulate (as best it can) an infinite inertia load. In carrying out the experiments it was necessary to disconnect the engine speed governor to prevent it from interacting with the operation of the dynamometer constant speed controller. Consequently, during these experiments the fuel rack position had to be demanded directly.

8.4.2.3 Response of the Engine to the Fuel Rack Test Signal

The test signal used to disturb the engine fuel rack input is shown in Figure 8.65; it results in a step change in the fuel rack position from 9.1 mm to 9.54 mm at 12 seconds, and step reduces it to 8.66 mm at 24 seconds. Figures 8.66 to 8.71 show how the fuel rack actuator, engine torque, engine speed, exhaust gas pressure, turbocharger speed and boost pressure respond to the test signal.

Figure 8.66 shows that the fuel rack actuator responds very rapidly to the test signal. The resulting changes in the quantity of fuel supplied to the engine give rise to the variation in load torque shown in Figure 8.67 and the variation in engine speed shown in Figure 8.68. It is evident that the ability of the dynamometer to simulate an infinite inertia load is not as good as might have been hoped for, since the engine speed varies significantly in response to relatively minor changes in engine torque. For example, when the position of the fuel rack actuator is changed at 24 seconds, the engine speed subsequently changes by some 30 rpm before the dynamometer is able to correct the speed error.

The speed perturbation is relatively unimportant for those engine responses which depend primarily on the response of the turbocharger; this is because the turbocharger response lasts for a significantly longer time than does the engine speed perturbation. However, the variation in engine speed is more serious when investigating the engine torque response, since if engine speed changes, some torque is used to accelerate the engine and this is excluded from the torque response (Figure 8.67), - which is a measure of the load torque only.

The change in fuel rack position also changes the pressure of the gas in the exhaust manifold (Figure 8.69), and this in turn results in changes to turbocharger speed (Figure 8.70) and boost pressure (Figure 8.71).

8.4.2.4 Response of the Engine to the Turbine Nozzle Test Signal

The test signal used to disturb the turbine nozzle restriction input is shown in Figure 8.72: it creates a step change in turbine restriction from 25% to 48.75% at 12 seconds and then at 24 seconds step reduces turbine restriction to 1.25%. Figures 8.73 to 8.76 show how the turbine nozzle actuator, exhaust gas pressure, boost pressure and engine torque respond to the test signal.

Figure 8.73 shows that the turbine nozzle actuator responds rapidly to the test signal. The change made to the turbine nozzle restriction was almost the maximum possible and, not surprisingly, it resulted in a major change to the pressure of the gas in the exhaust manifold (Figure 8.74). In turn boost pressure changed, as shown in Figure 8.75. At the operating condition at which the engine was tested, an increase in turbine restriction reduced engine torque (and vice versa), as is shown in Figure 8.76.

8.4.3 Comparison of the Simulated and Experimental Engine Responses

The response of the engine model (Section 8.4.1) and the response of the real engine (Section 8.4.2.) to the fuel rack and turbine nozzle test signals were not measured at exactly the same engine operating condition (Table 8.6), or even using the same test signals (Table 8.7). Additional computer simulations could, of course, have been carried out, to repeat the experiments made on the real engine. However, this was not considered to be worthwhile, because the correct turbine and fuel pump characteristics were not

available (see Chapter 4), and therefore the results could not have been used in a quantitative comparison. In spite of these differences the nature of the experiments is similar, and a worthwhile qualitative comparison of the responses can still be made.

The engine responses measured during the engine experiments and the equivalent responses obtained using the engine model are listed in Table 8.8. A comparison of the respective time histories, shows that the model correctly predicts the behaviour of dominant engine responses, such as the relationship between rack position and engine torque, and also successfully predicts the response of much weaker dynamic interactions, such as the interaction between turbine nozzle restriction and engine torque.

8.5 Summary

This chapter has described the experiments carried out using the engine simulator, to test the simulator software. The results presented in Section 8.2 show that the engine model predicts gas responses in the engine cylinders and manifolds, typical of those measured on real engines. The results presented in Section 8.3, show that the steady state performance of the engine model generally exhibits the same trends as does the real engine and with an accuracy generally within 10% of the real engine, - and this, without making any special effort to tune the engine model. Finally, the results presented in Section 8.4 show that the engine model correctly predicts the dynamic behaviour of the engine to changes in engine controls. It is therefore inconceivable that

errors remain in the computer model coding, which would seriously effect the model execution time results which are presented in Chapter 9.

8.6 References

8.1 N Watson, M Marzouk.
A Non-linear Digital Simulation of Turbocharged Diesel Engines Under Transient Conditions.
1977, SAE 770123.

8.2 M Marzouk, N Watson.
Load Acceptance of Turbocharged Diesel Engines.
1978, I Mech E Paper C54/78.

8.3 N Watson.
Transient Performance Simulation and Analysis of Turbocharged Diesel Engines.
1981, SAE 810338.

8.4 E W Roberts.
Variable Geometry Turbocharging, Optimisation and Control.
1984, PhD Thesis, University of Bath.

8.5 A D Jones.
Control of a Diesel Engine Fitted with a Variable Geometry Turbocharger (Transfer Report).
1986, University of Bath.

8.7 Tables

		Engine Speed (rpm)							
		695	894	1106	1293	1503	1698	1905	2123
Fuel Mass/Shot	(grams)	0.123	0.131	0.129	0.128	0.125	0.123	0.121	0.117
Bmep	(bar)	9.09	10.31	11.0	11.21	11.3	10.88	10.44	9.62
Sfc	(g/kwhr)	263.0	247.0	227.0	221.0	215.0	219.0	225.0	236.0
Brake Power	(kw)	58.4	85.3	112.8	134.2	157.2	171	184.0	189.0
Boost Pressure	(bar)	1.23	1.35	1.49	1.58	1.74	1.85	2.0	2.06
Inlet Manifold Temperature	(k)	325.0	334.0	342.0	349.0	362.0	370.0	383.0	390.0
Turbocharger Speed $*10^3$	(rpm)	42.6	47.8	55.4	59.0	68.4	73.5	80.7	84.8
Exhaust Manifold Pressure	(bar)	1.12	1.19	1.26	1.38	1.49	1.62	1.78	1.92
Turbine Inlet Temperature	(k)	851.0	870.0	875.0	875.0	870.0	870.0	871.0	875.0
Engine Air Mass Flow	(cfm)	132.0	191.0	251.0	308.0	381.0	443.0	520.0	575.0

Table 8.1a Steady State Performance of Engine Model - 0% Turbine Restriction

		Engine Speed (rpm)							
		695	894	1106	1293	1503	1698	1905	2123
Fuel Mass/Shot	(grams)	0.123	0.131	0.129	0.128	0.125	0.123	0.121	0.117
Bmep	(bar)	9.10	10.15	10.77	11.23	11.35	10.92	10.36	9.63
Sfc	(g/kwhr)	263.8	251.5	233.6	222.0	215.1	219.1	227.7	236.2
Brake Power	(kw)	58.6	84.0	110.6	134.6	158.1	171.8	182.9	189.4
Boost Pressure	(bar)	1.21	1.33	1.44	1.57	1.74	1.87	2.05	2.15
Inlet Manifold Temperature	(k)	319.8	330.3	338.8	350.8	361.1	377.8	391.8	399.2
Turbocharger Speed $*10^3$	(rpm)	35.0	45.2	52.3	59.7	68.7	74.9	83.5	89.3
Exhaust Manifold Pressure	(bar)	1.13	1.22	1.3	1.41	1.57	1.73	1.94	2.13
Turbine Inlet Temperature	(k)	914.0	978.0	989.0	986.0	945.0	944.0	951.0	950.0
Engine Air Mass Flow	(cfm)	120.4	172.6	224.7	285.0	360.3	420.8	501.4	568.6

Table 8.1b Steady State Performance of Experimental TL11 Engine - 0% Turbine Restriction

		Engine Speed (rpm)					
		1106	1323	1507	1702	1898	2032
Fuel Mass/Shot	(grams)	0.129	0.128	0.124	0.122	0.121	0.119
Bmep	(bar)	11.13	11.36	11.19	10.7	10.26	9.81
Sfc	(g/kwhr)	225.3	219.0	215.0	222.0	229.0	235.8
Brake Power	(kw)	114.0	139.08	156.0	168.5	180.2	184.6
Boost Pressure	(bar)	1.57	1.75	1.92	2.07	2.21	2.35
Inlet Manifold Temperature	(k)	348.0	362.0	374.0	386.0	398.0	412.0
Turbocharger Speed $*10^3$	(rpm)	58.2	67.6	75.0	81.0	86.6	92.8
Exhaust Manifold Pressure	(bar)	1.37	1.56	1.73	1.94	2.18	2.35
Turbine Inlet Temperature	(k)	846.0	850.0	850.0	850.0	865.0	870.0
Engine Air Mass Flow	(cfm)	262.0	339.0	407.0	480.0	550.0	605.0

Table 8.2a Steady State Performance of Engine Model - 25% Turbine Restriction

		Engine Speed (rpm)					
		1106	1323	1507	1702	1898	2032
Fuel Mass/Shot	(grams)	0.129	0.128	0.124	0.122	0.121	0.119
Bmep	(bar)	10.93	11.27	11.22	10.87	10.31	9.82
Sfc	(g/kwhr)	229.4	221.5	214.9	219.6	229.2	235.6
Brake Power	(kw)	112.0	138.1	156.7	171.4	181.3	184.9
Boost Pressure	(bar)	1.52	1.69	1.89	2.08	2.30	2.35
Inlet Manifold Temperature	(k)	349.2	364.3	378.6	393.8	409.8	416.3
Turbocharger Speed $*10^3$	(rpm)	56.8	65.9	74.6	82.64	90.9	94.4
Exhaust Manifold Pressure	(bar)	1.39	1.55	1.76	1.99	2.29	2.38
Turbine Inlet Temperature	(k)	982.0	960.0	935.0	928.0	942.0	950.0
Engine Air Mass Flow	(cfm)	233.6	300.1	375.3	452.9	552.2	574.9

Table 8.2b Steady State Performance of Experimental TL11 Engine - 25% Turbine Restriction

		Engine Speed (rpm)					
		700	890	1096	1310	1500	1702
Fuel Mass/Shot	(grams)	0.119	0.127	0.126	0.124	0.122	0.118
Bmep	(bar)	8.86	10.22	10.89	10.93	10.94	10.16
Sfc	(g/kwhr)	261.0	241.0	224.0	220.0	216.0	225.0
Brake Power	(kw)	57.4	84.12	110.5	132.5	151.9	160.0
Boost Pressure	(bar)	1.12	1.38	1.56	1.75	1.98	2.17
Inlet Manifold Temperature	(k)	320.0	336.0	347.0	361.0	378.0	392.0
Turbocharger Speed $*10^3$	(rpm)	32.0	50.0	58.0	67.5	76.7	84.0
Exhaust Manifold Pressure	(bar)	1.16	1.3	1.45	1.66	1.92	2.2
Turbine Inlet Temperature	(k)	880.0	860.0	855.0	853.0	855.0	857.0
Engine Air Mass Flow	(cfm)	129.0	193.0	259.0	334.0	410.0	458.0

Table 8.3a Steady State Performance of Engine Model - 40% Turbine Restriction

		Engine Speed (rpm)					
		700	890	1096	1310	1500	1702
Fuel Mass/Shot	(grams)	0.119	0.127	0.126	0.124	0.122	0.118
Bmep	(bar)	8.93	10.13	10.95	11.3	11.19	10.65
Sfc	(g/kwhr)	259.7	244.6	223.8	213.9	211.9	216.9
Brake Power	(kw)	57.9	83.5	111.2	137.2	155.5	167.9
Boost Pressure	(bar)	1.28	1.45	1.61	1.82	2.04	2.25
Inlet Manifold Temperature	(k)	328.8	345.0	358.3	374.8	391.0	407.0
Turbocharger Speed $*10^3$	(rpm)	39.2	51.0	60.3	69.9	79.2	87.27
Exhaust Manifold Pressure	(bar)	1.25	1.32	1.48	1.69	1.95	2.24
Turbine Inlet Temperature	(k)	984.0	1015.0	1006.0	982.0	960.0	955.0
Engine Air Mass Flow	(cfm)	124.9	178.9	242.1	315.9	393.7	477.9

Table 8.3b Steady State Performance of Experimental TL11 Engine - 40% Turbine Restriction

		Engine Speed (rpm)				
		705	899	1105	1302	1501
Fuel Mass/Shot	(grams)	0.124	0.132	0.13	0.128	0.125
Bmep	(bar)	9.65	10.83	11.15	11.25	11.24
Sfc	(g/kwhr)	249.0	237.0	226.7	221.1	216.0
Brake Power	(kw)	62.94	90.1	114.0	135.6	156.0
Boost Pressure	(bar)	1.22	1.44	1.75	1.88	2.2
Inlet Manifold Temperature	(k)	324.0	339.0	360.0	371.0	393.0
Turbocharger Speed $*10^3$	(rpm)	41.0	52.0	66.4	72.7	83.5
Exhaust Manifold Pressure	(bar)	1.2	1.4	1.64	1.88	2.29
Turbine Inlet Temperature	(k)	875.0	891.0	870.0	865.0	860.0
Engine Air Mass Flow	(cfm)	141.0	202.0	283.0	349.0	440.0

Table 8.4a Steady State Performance of Engine Model - 50% Turbine Restriction

		Engine Speed (rpm)				
		705	899	1105	1302	1501
Fuel Mass/Shot	(grams)	0.124	0.132	0.13	0.128	0.125
Bmep	(bar)	9.49	10.93	11.56	11.63	11.19
Sfc	(g/kwhr)	253.7	234.7	218.6	214.4	217.4
Brake Power	(kw)	62.0	91.0	118.4	140.3	155.6
Boost Pressure	(bar)	1.3	1.53	1.75	1.99	2.24
Inlet Manifold Temperature	(k)	326.2	346.8	371.2	380.8	398.8
Turbocharger Speed $*10^3$	(rpm)	41.93	56.7	67.3	76.97	86.2
Exhaust Manifold Pressure	(bar)	1.25	1.45	1.7	1.99	2.34
Turbine Inlet Temperature	(k)	915.0	958.0	933.0	915.0	902.0
Engine Air Mass Flow	(cfm)	131.1	191.6	262.5	338.7	420.8

Table 8.4b Steady State Performance of Experimental TL11 Engine - 50% Turbine Restriction

Processor:	Motorola MC68000 (8MHZ)
RAM:	3/4 Mbyte (one wait state)
Mass Storage:	Two, 1.2 Mbyte 8 inch floppy disc drives
Graphics:	EFCIS graphics board - 512x512 pixels, 8 colours
I/O:	Analogue card - 4, 12 bit DAC - 8 multiplexed 12 bit ADC
Operating System:	TRIPOS

Table 8.5 Computer System Decsription.

	Simulator Operating Condition	Engine Operating Condition
Engine Speed	1500 rpm	1500 rpm
Nominal Torque	680 Nm	500 Nm
Load System	"infinite" inertia	dynamometer "constant speed" mode
Static Timing	27 btdc	22 btdc
Nominal Turbine Nozzle Restriction	25%	25%

Table 8.6 Comparison of Engine Simulator and Engine Operating Conditions.

		Fuel Rack Test Signal			Turbine Nozzle Restriction Test Signal		
		Nominal signal level (mm)	Amplitude (mm)	Period (seconds)	Nominal signal level (%)	Amplitude (%)	Period (seconds)
Simulator Experiments	Experiment 1	11.0	1.0	16.0	25.0	10.0	16.0
	Experiment 2	11.0	2.0	16.0	25.0	20.0	16.0
Engine Experiments		9.1	0.44	24.0	25.0	23.75	24.0

Table 8.7 Comparison of Disturbance Signals used for Testing the Engine Simulator and the Real Engine.

Response	Rack Response Experiments		Turbine Nozzle Restriction Experiments	
	Simulator Response (Fig No)	Engine Response (Fig No)	Simulator Response (Fig No)	Engine Response (Fig No)
Fuel Rack Actuator	8.47	8.66	-	-
Turbine Nozzle Actuator	-	-	8.56	8.73
Engine Speed	-	8.68	-	-
Engine Torque	8.48	8.67	8.57,8.58	8.76
Exhaust Manifold Gas Rressure	8.49	8.69	8.59	8.74
Exhaust Manifold Gas Temperature	8.50	-	8.60	-
Exhaust Manifold Gas Fuel/Air Ratio	8.51	-	-	-
Turbocharger Speed	8.52	8.70	8.61	-
Inlet Manifold Gas Pressure	8.53	8.71	8.62	8.75
Inlet Manifold Gas Temperature	8.54	-	8.63	-

Table 8.8 Comparison of Engine Simulator and Engine Responses (Figure numbers)

Fig. 8.1 Rate of change of gas temperature

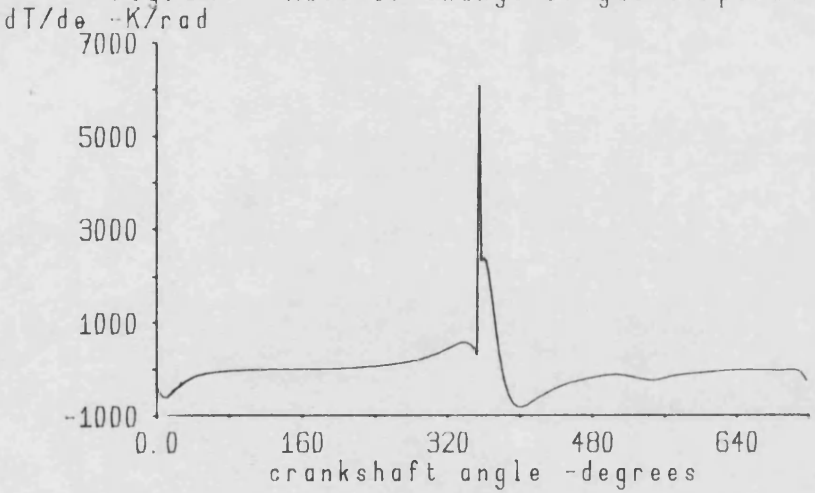


Fig. 8.2 Gas temperature response

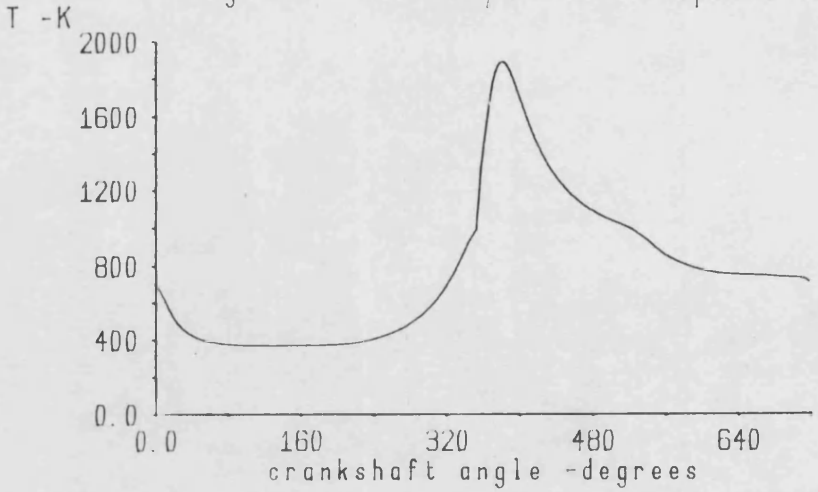


Fig. 8.3 Rate of change of fuel air ratio

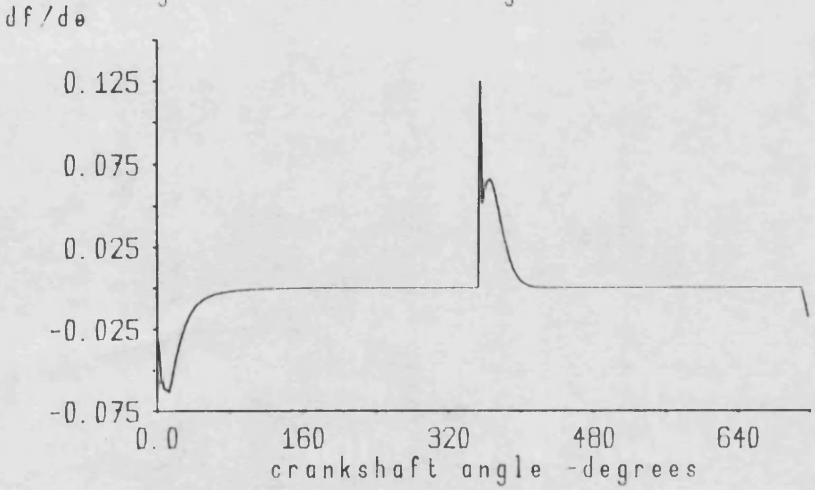


Fig. 8.4 Gas fuel air ratio response

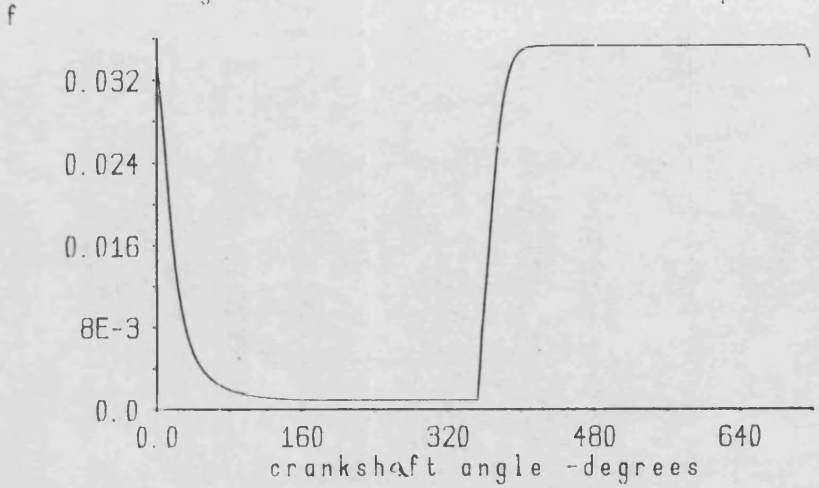


Fig. 85 Rate of change of gas mass

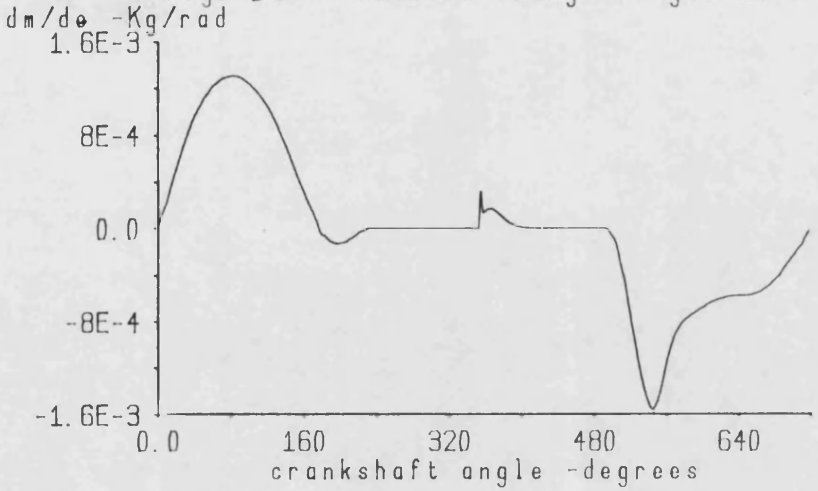


Fig . 86 Gas mass response

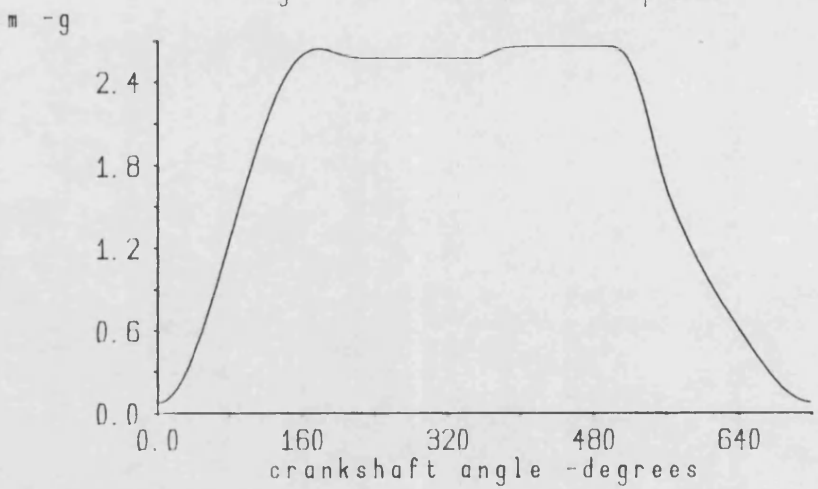


Fig. 87 Gas pressure response

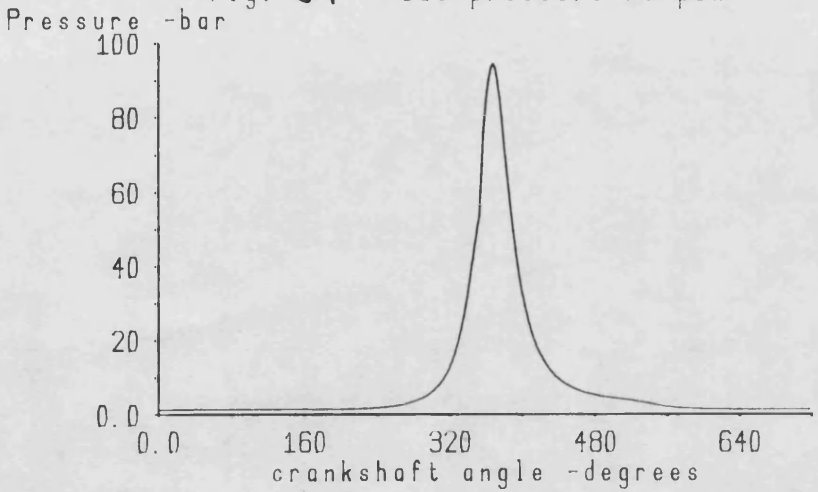


Fig. 88 Gas heat transfer response

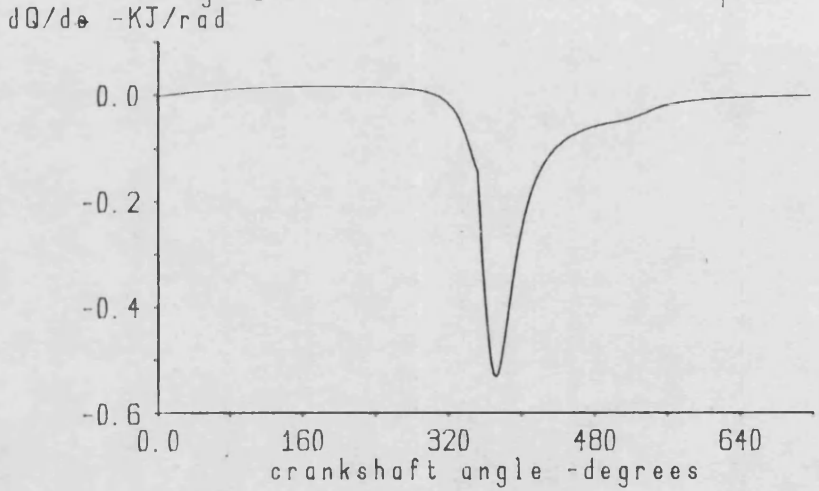


Fig. 8.9 Gas work response

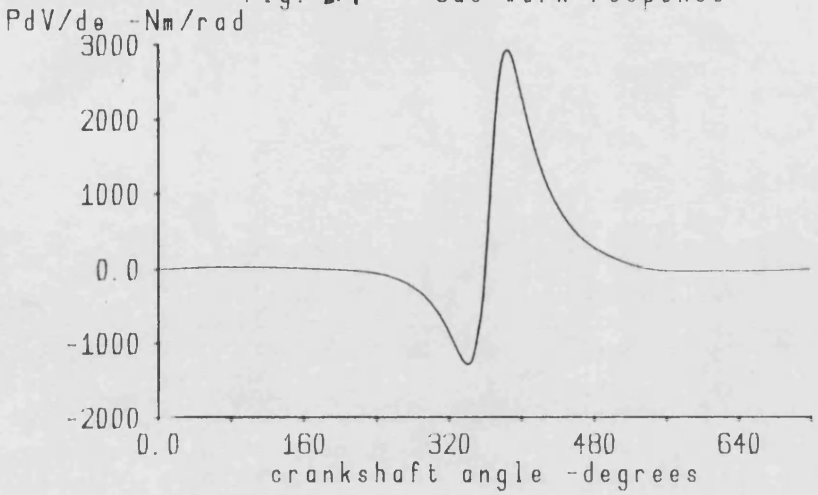


Fig. 8.10 Power cycle P-V diagram

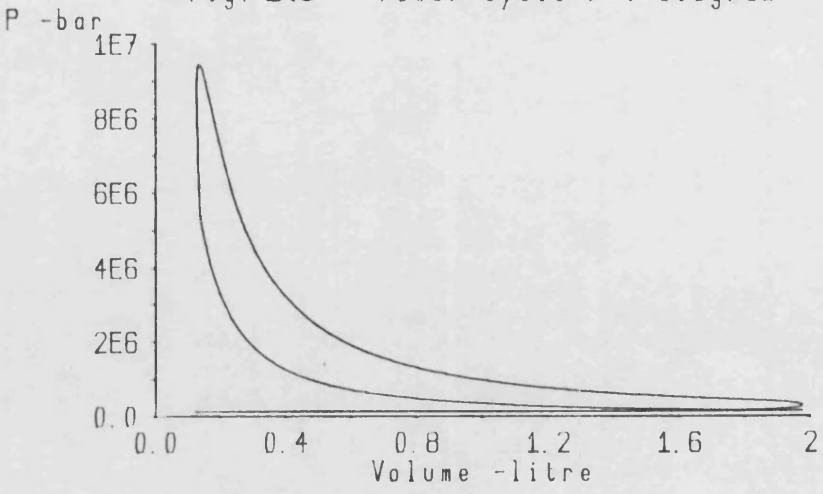


Fig. 8.11 Gas pressure response

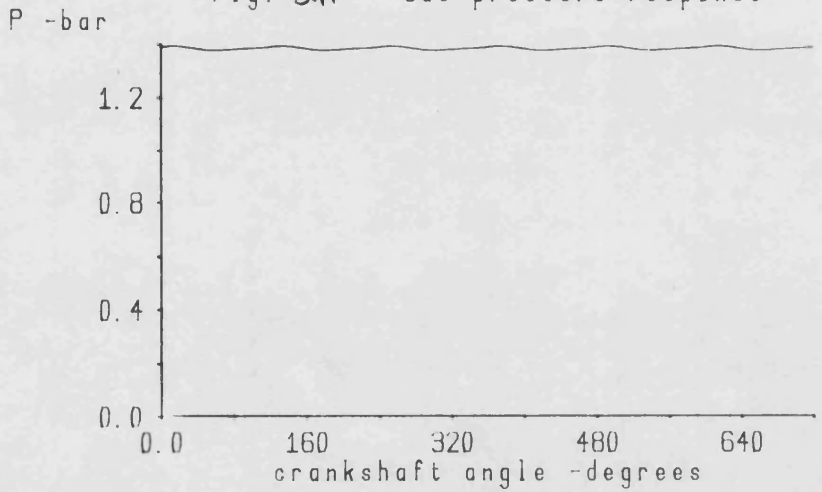


Fig. 8.12 Gas temperature response

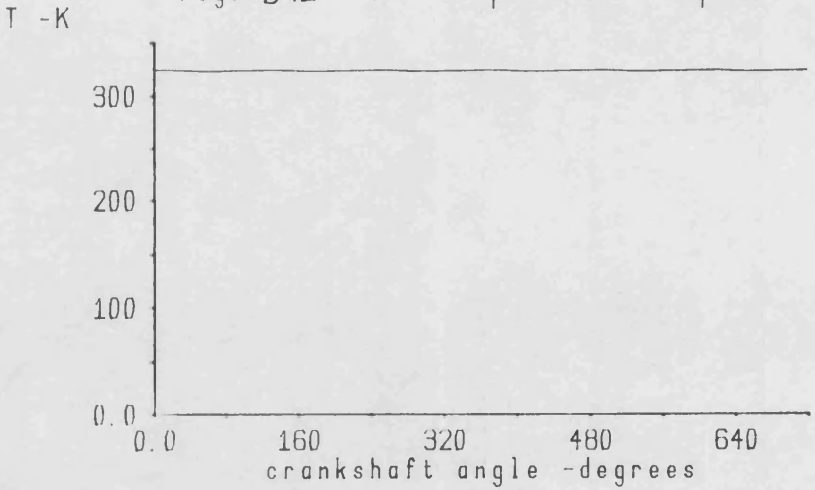


Fig. 8.13 Gas mass response

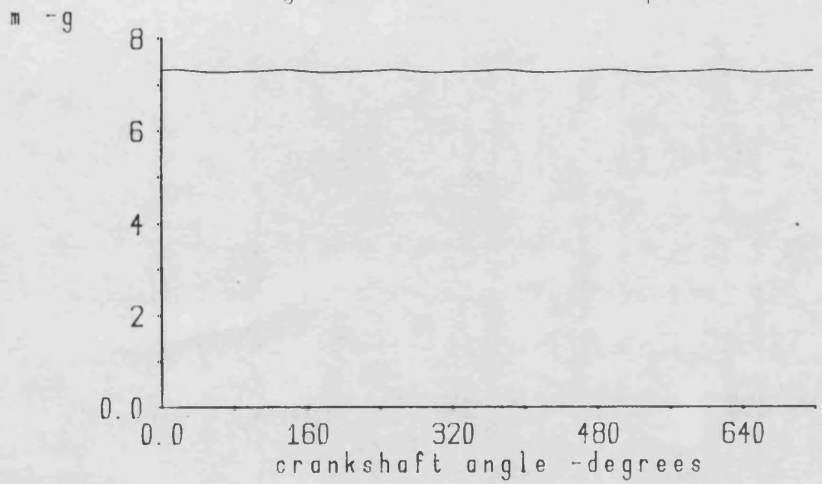


Fig. 8.14 Gas pressure response

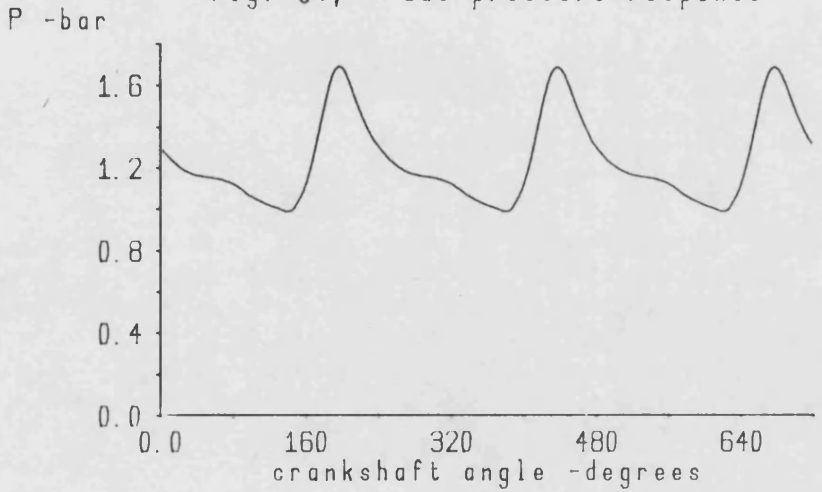


Fig. 8.15 Gas temperature response

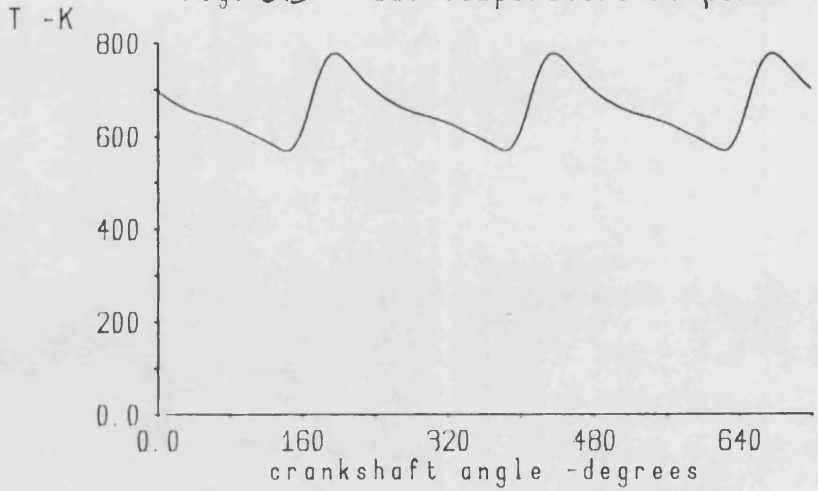


Fig. 8.16 Gas fuel air response

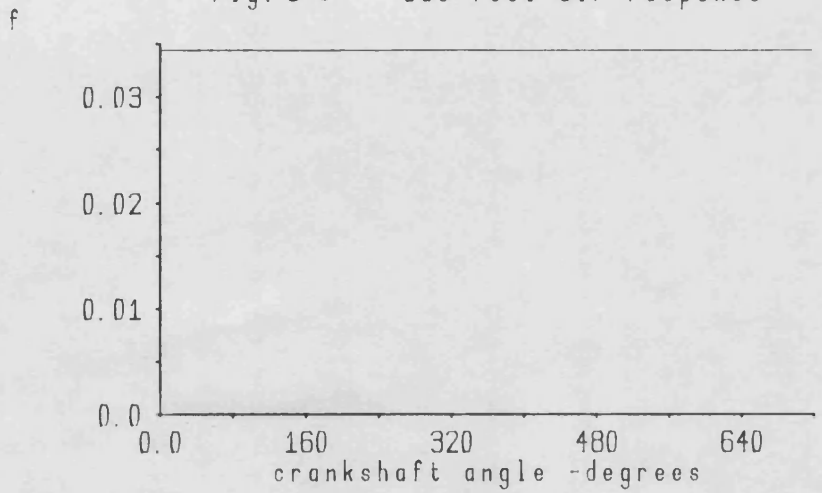


Fig. 8.17 Gas mass response

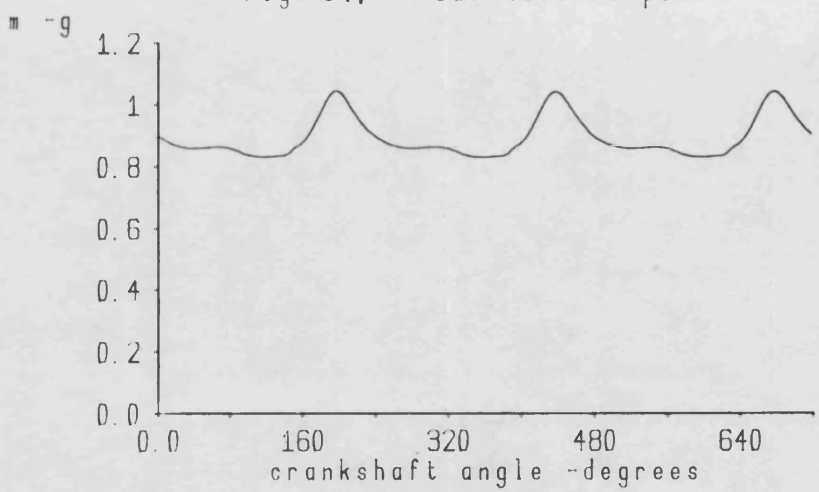


Fig. 8.18 Fuel rack response

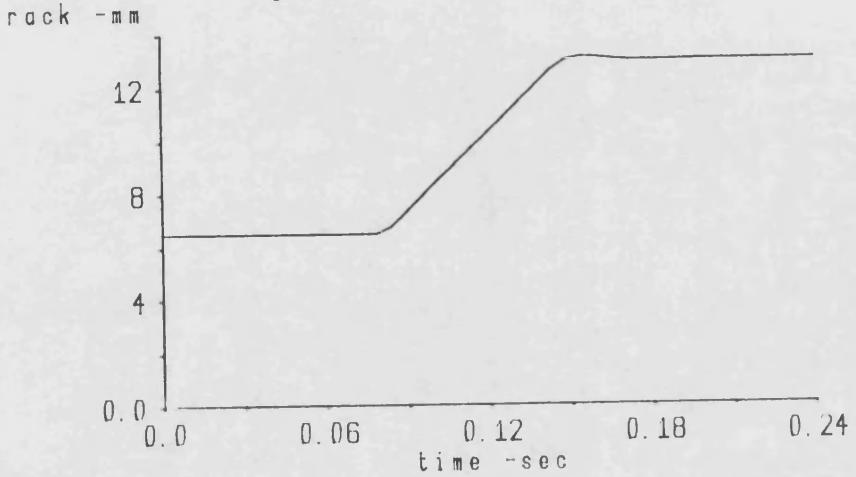


Fig. 8.19 Gas pressure response

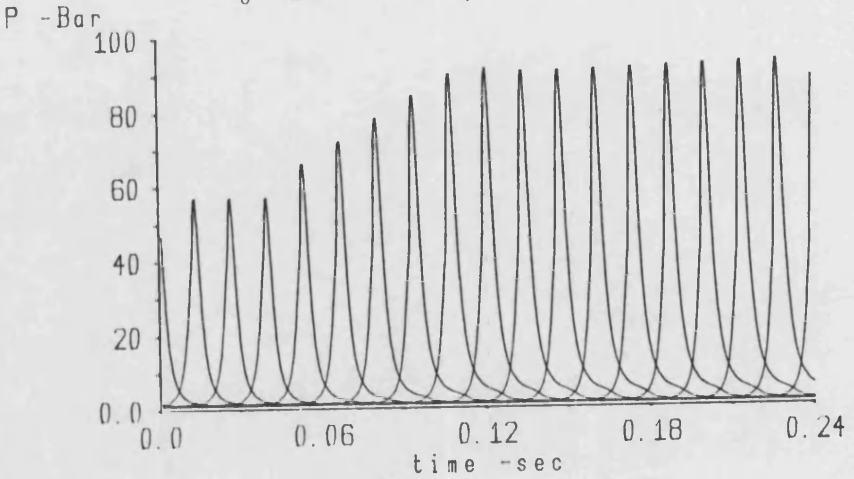


Fig. 8.20 Gas temperature response

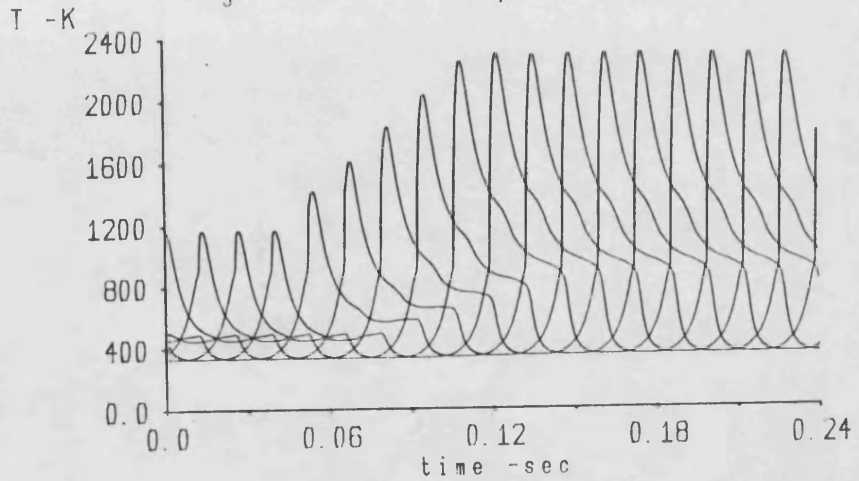


Fig. 8.21 Gas fuel-air response

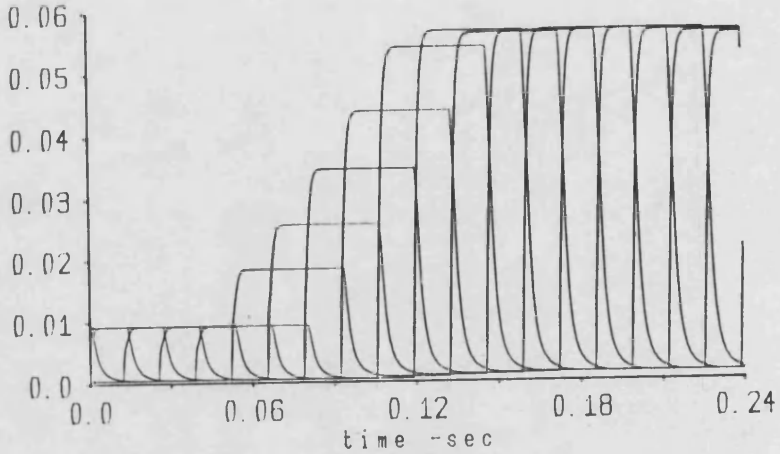


Fig. 8.22 Gas mass response

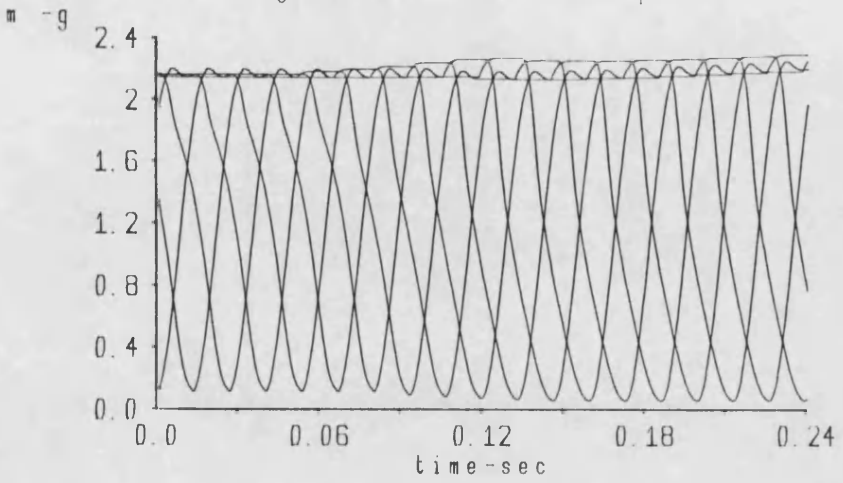


Fig. 8.23 Torque response

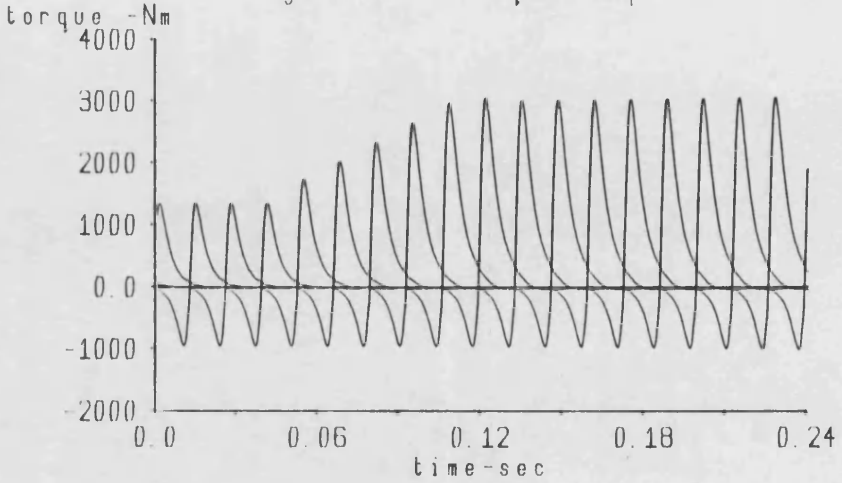


Fig. 3.24 Fuel timing response

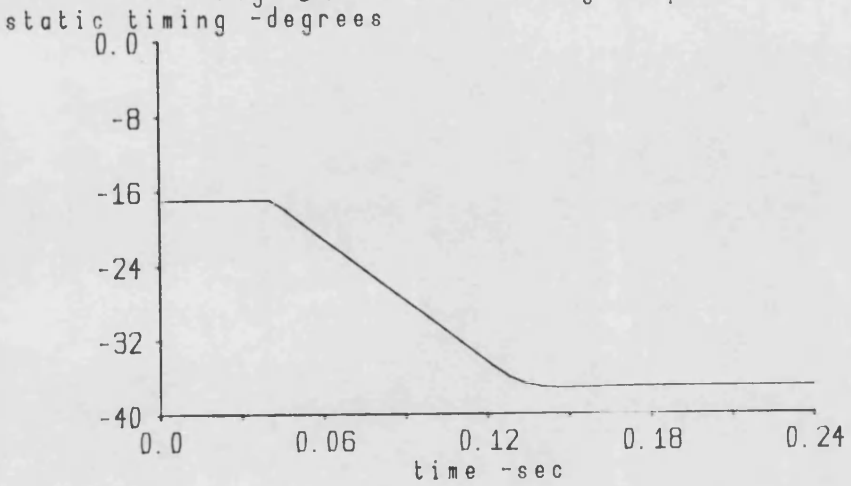


Fig. 3.25 Gas pressure response

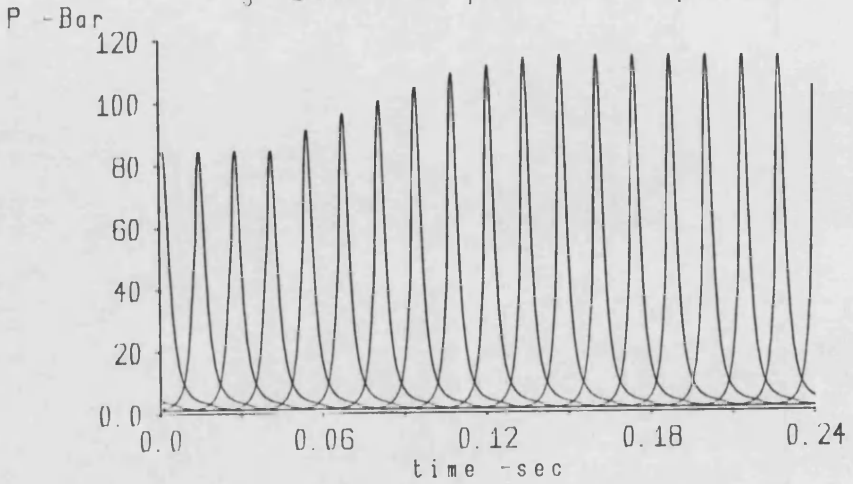


Fig. 3.26 Gas temperature response

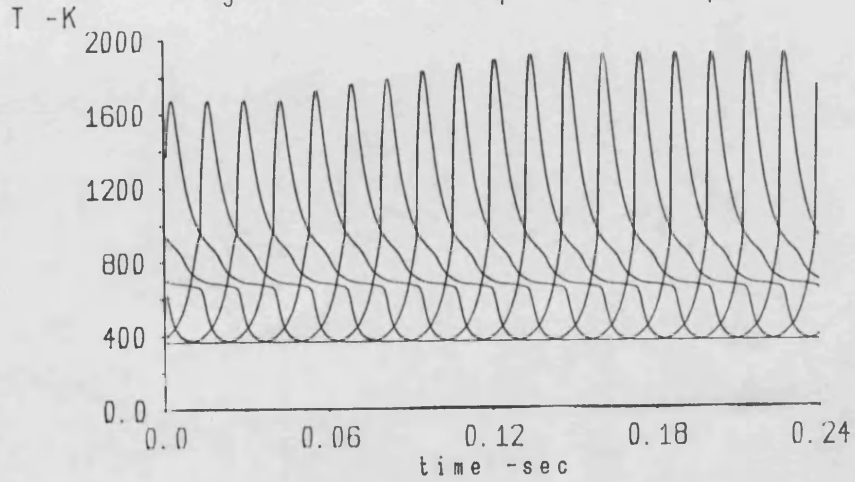


Fig. 3.27 Gas fuel-air response

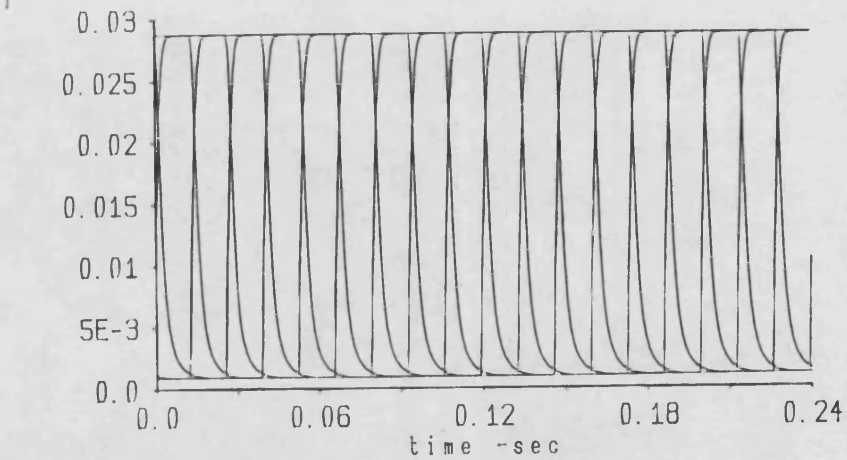


Fig. 328 Gas mass response

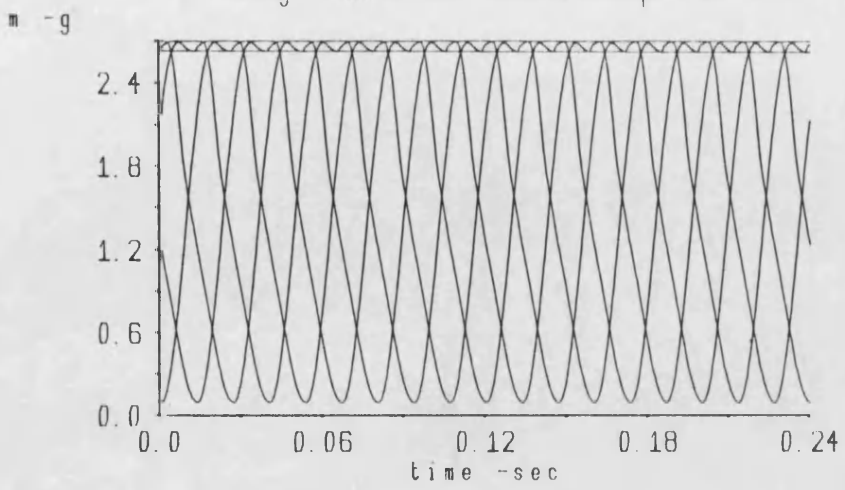


Fig. 329 Torque response

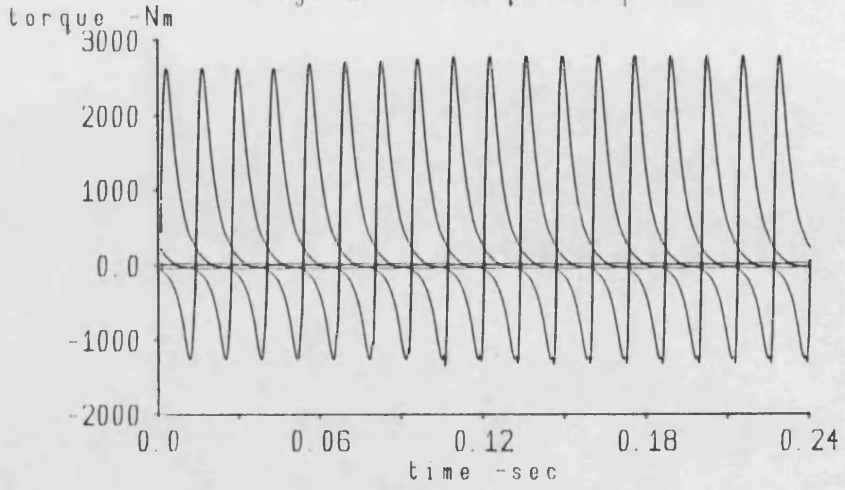


Fig. 8.30 Turbine restriction response

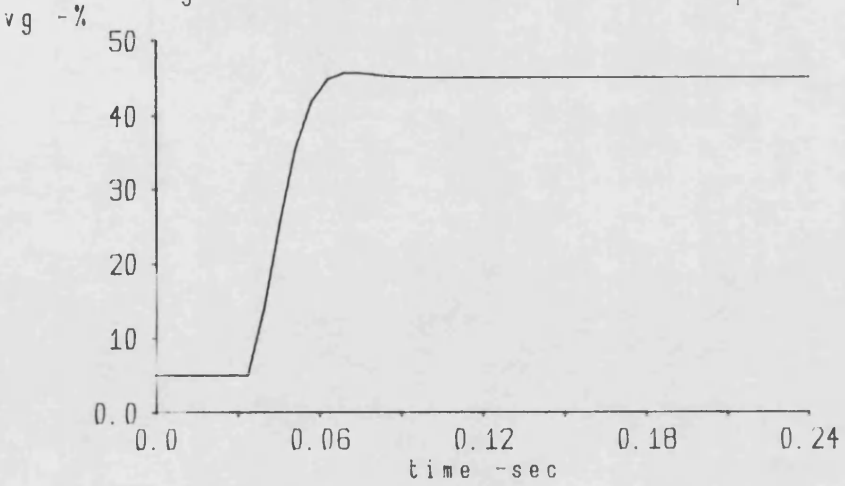


Fig. 8.31 Gas pressure response

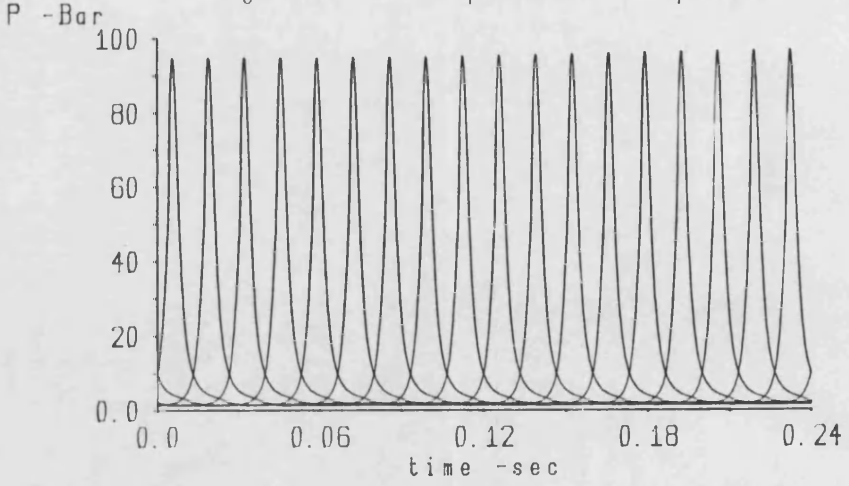


Fig. 8.32 Gas temperature response

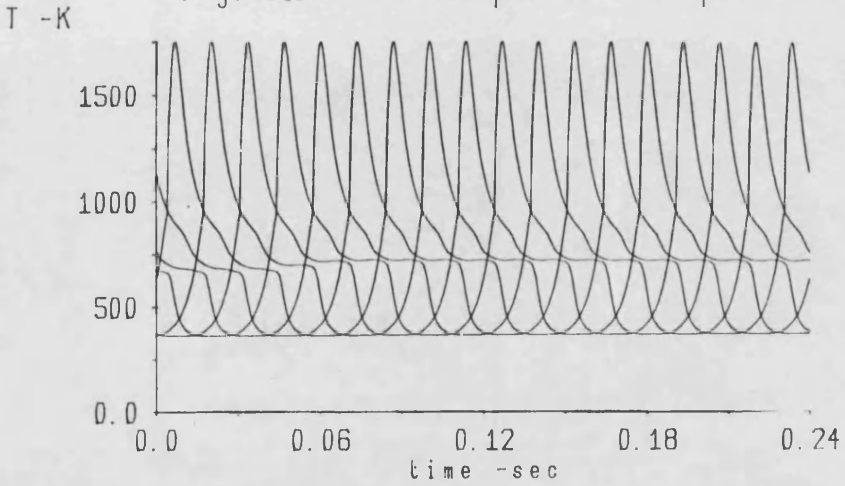


Fig. 8.33 Gas fuel-air response

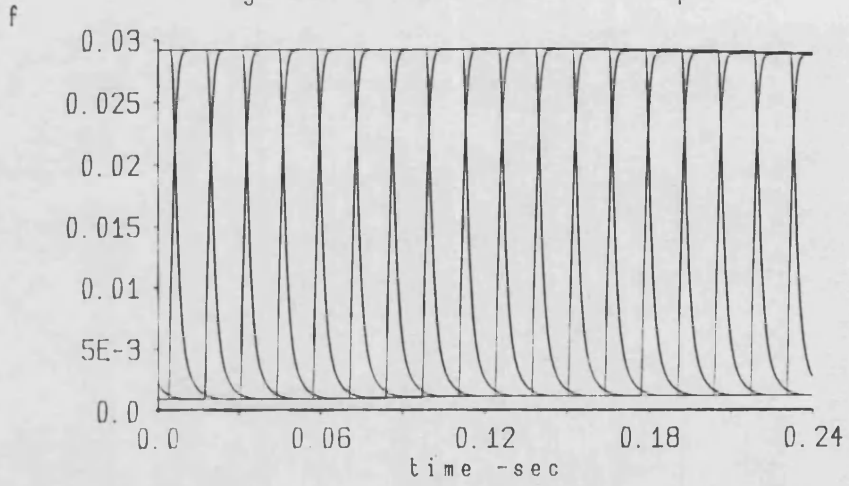


Fig. 8.34 Gas mass response

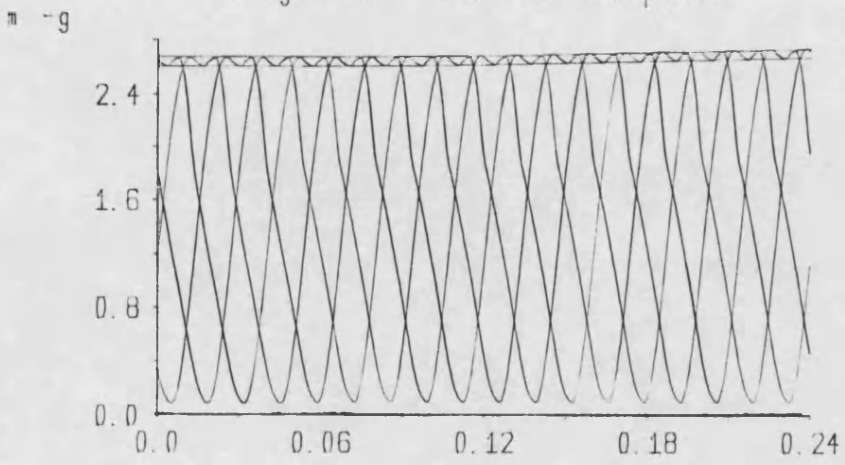
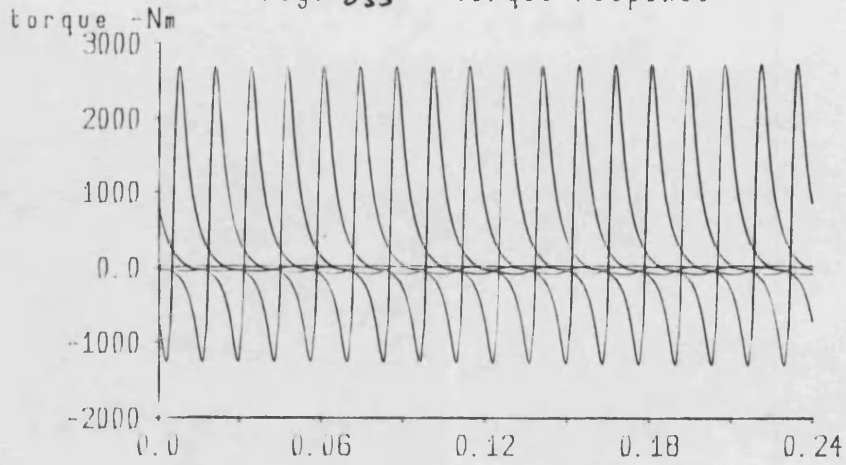


Fig. 8.35 Torque response



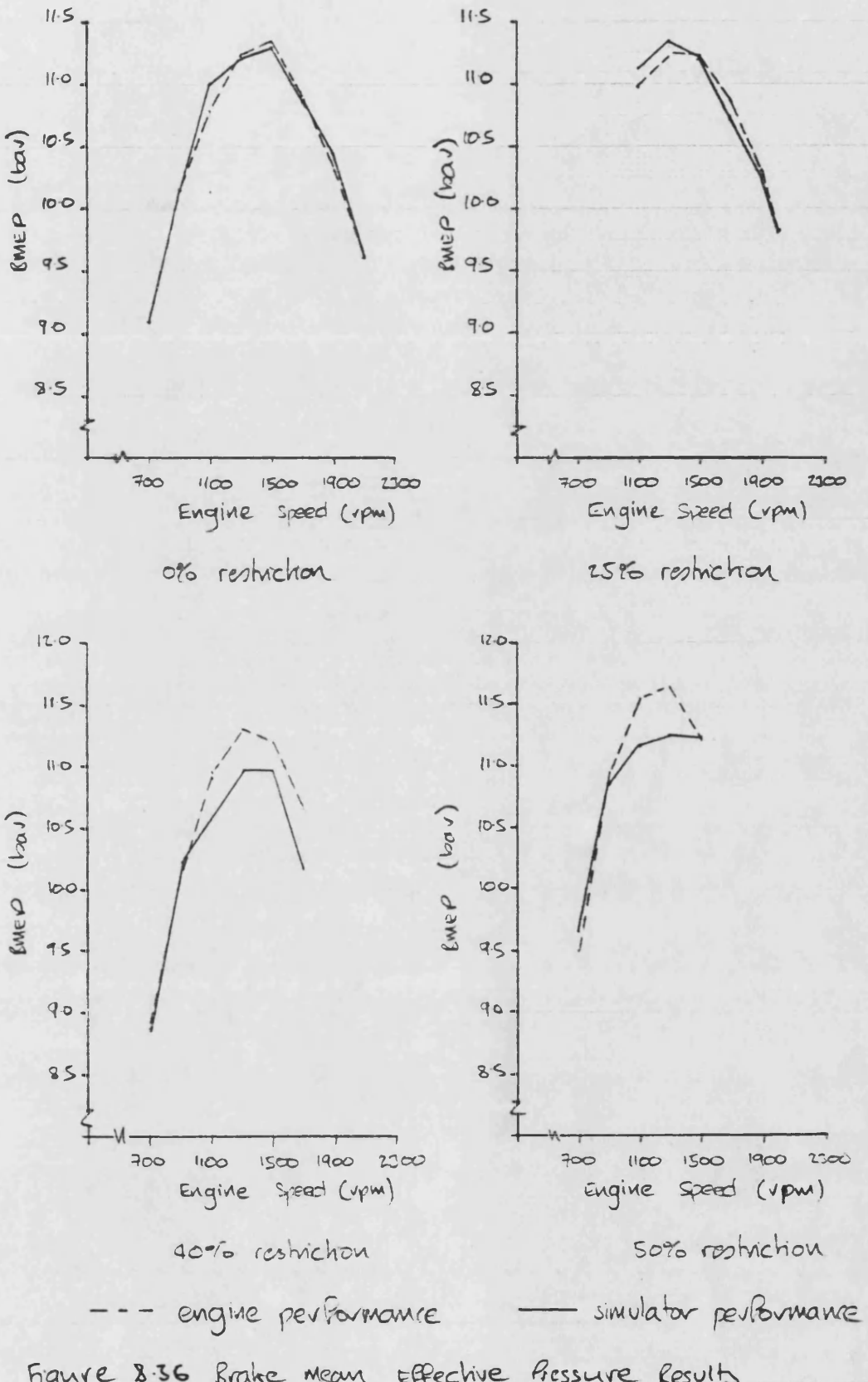


Figure 8.36 Brake mean effective pressure results

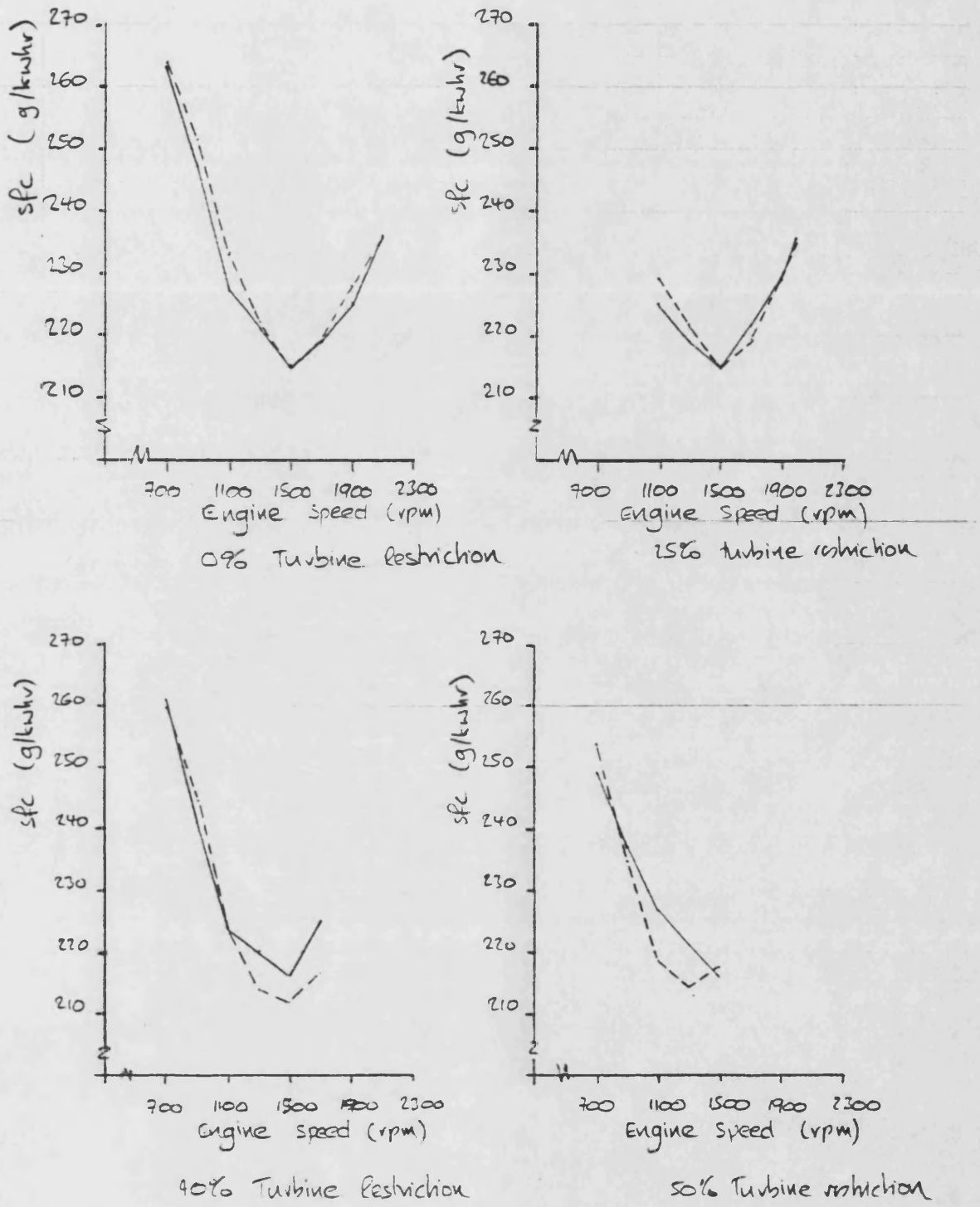


Figure 8.37 Specific Fuel Consumption Results

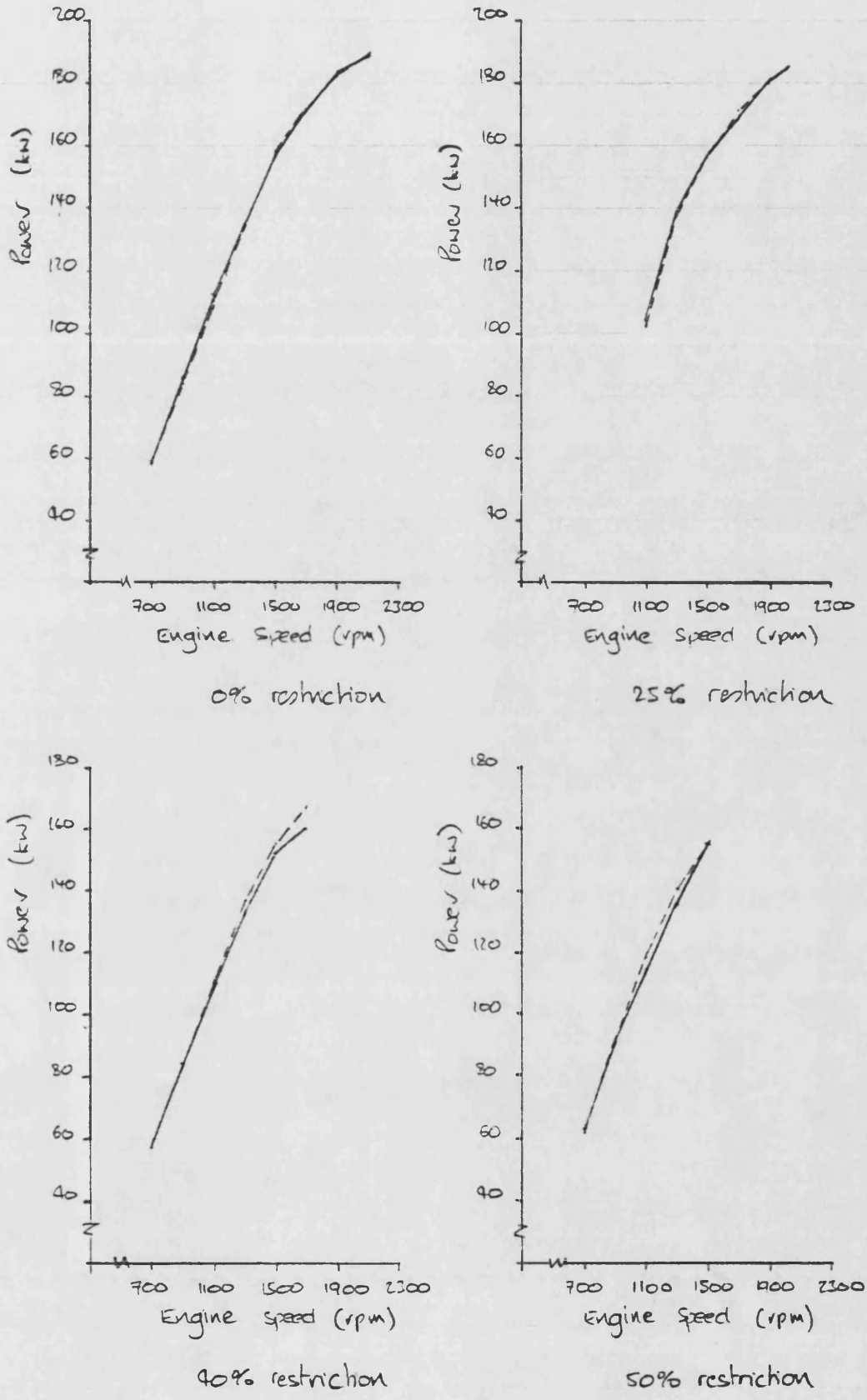


Figure 8.38 Engine Brake Power Results.

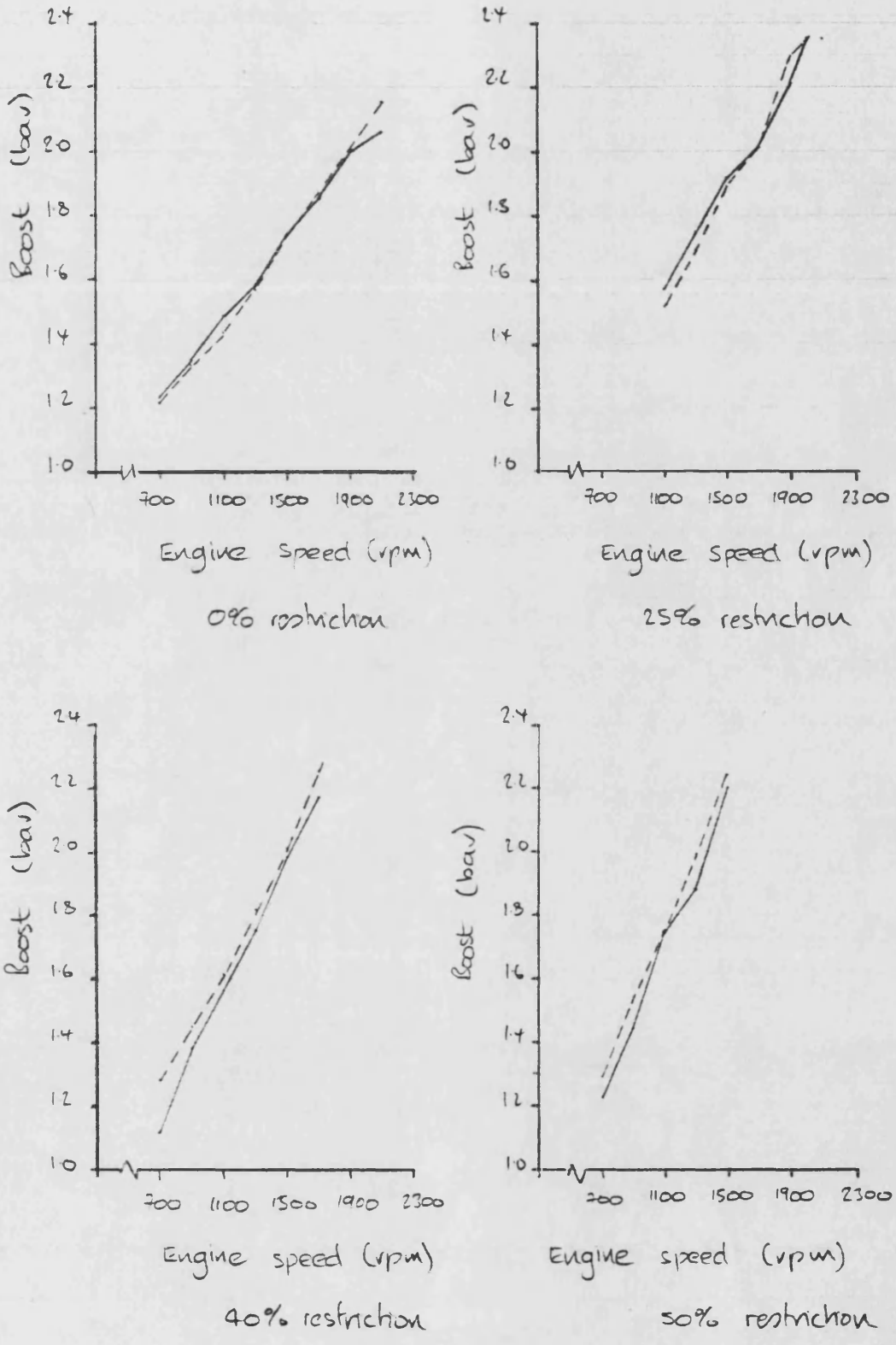


Figure 839 Boost Pressure Results

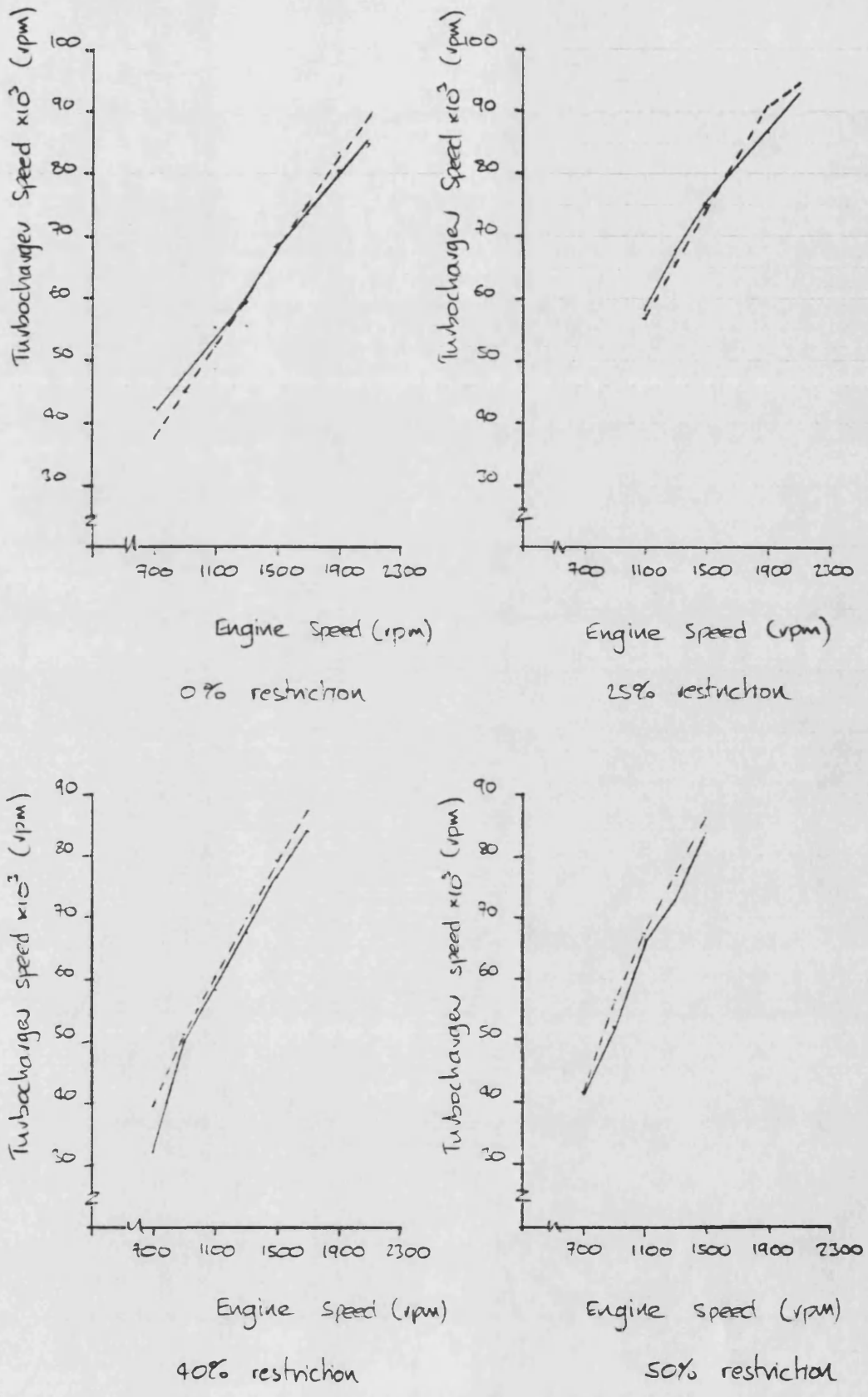


Figure 8.40 Turbocharger Speed Results

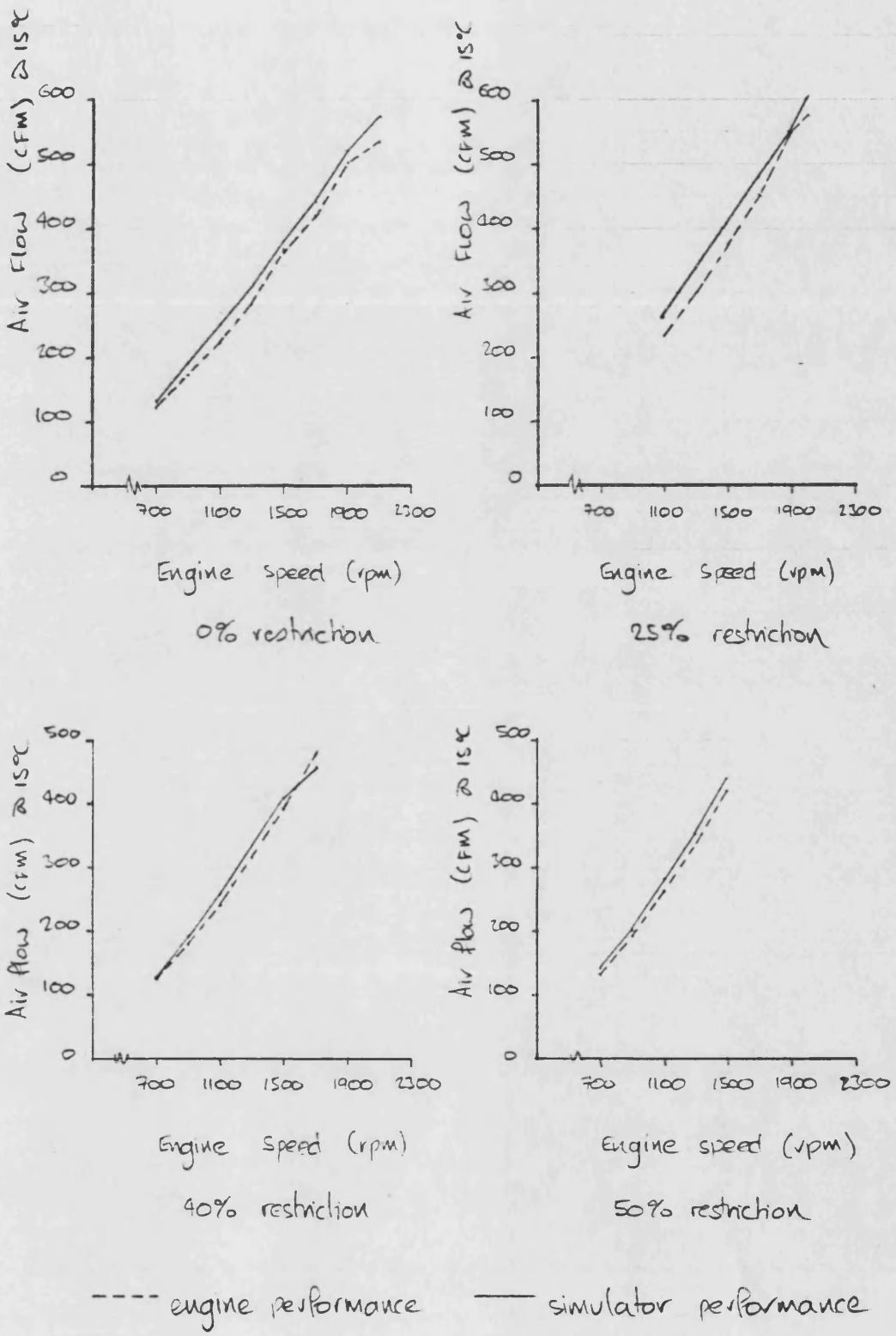


Figure 8.41 Air Flow Results

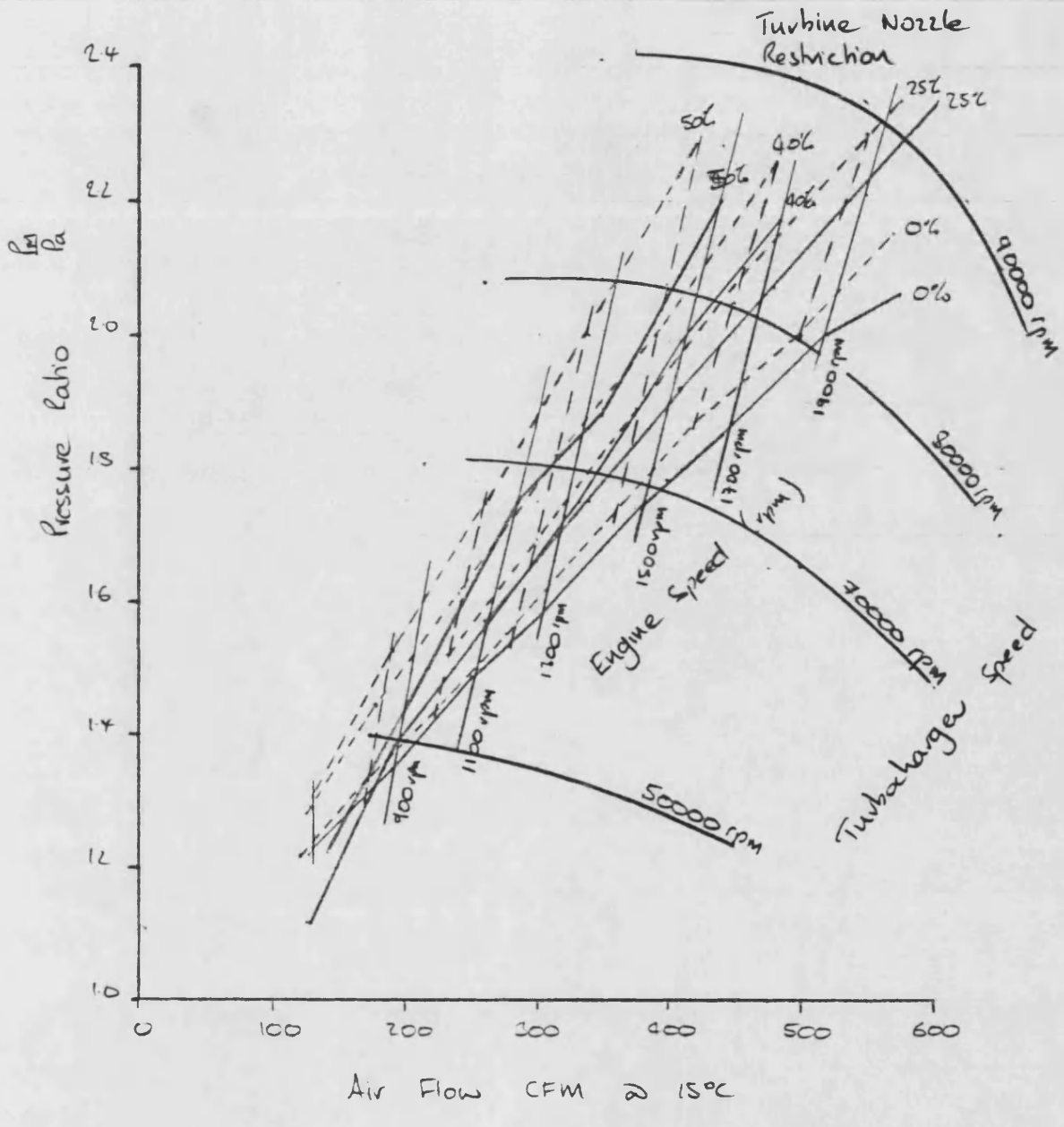


Figure 8.4-2 Compressor Performance

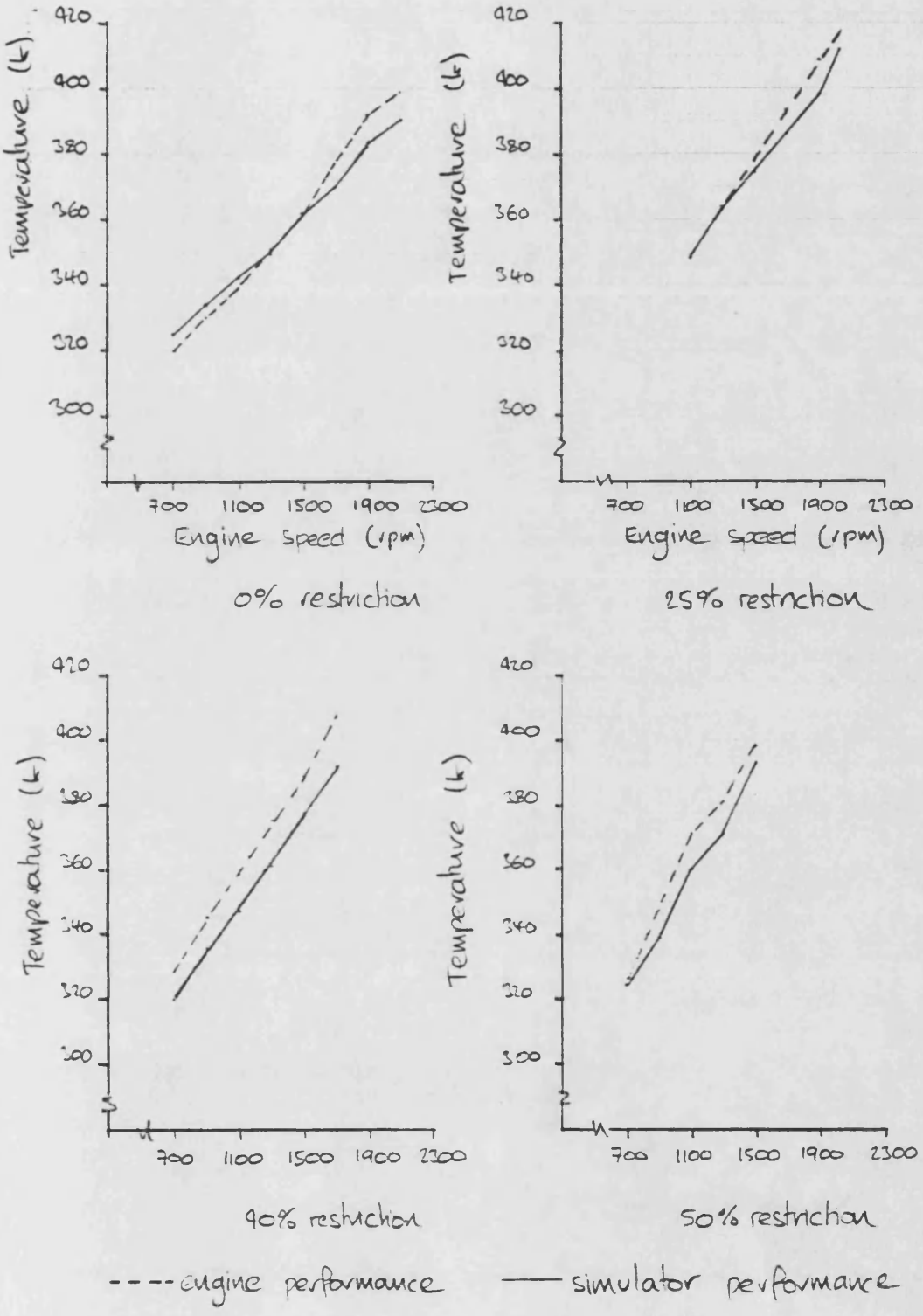


Figure 8.43 Inlet manifold temperature results

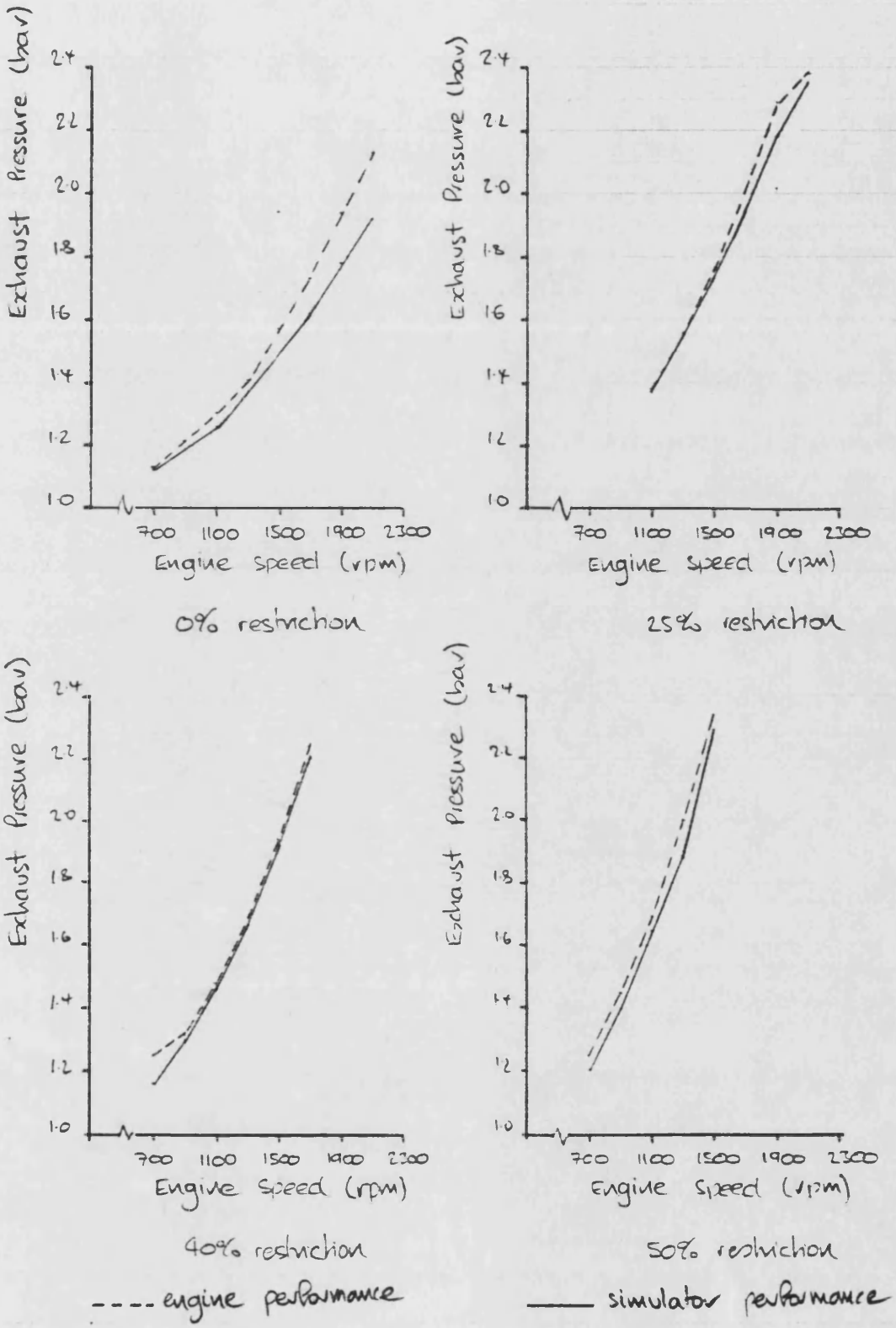


Figure 844 Exhaust manifold Gas Pressure Results.

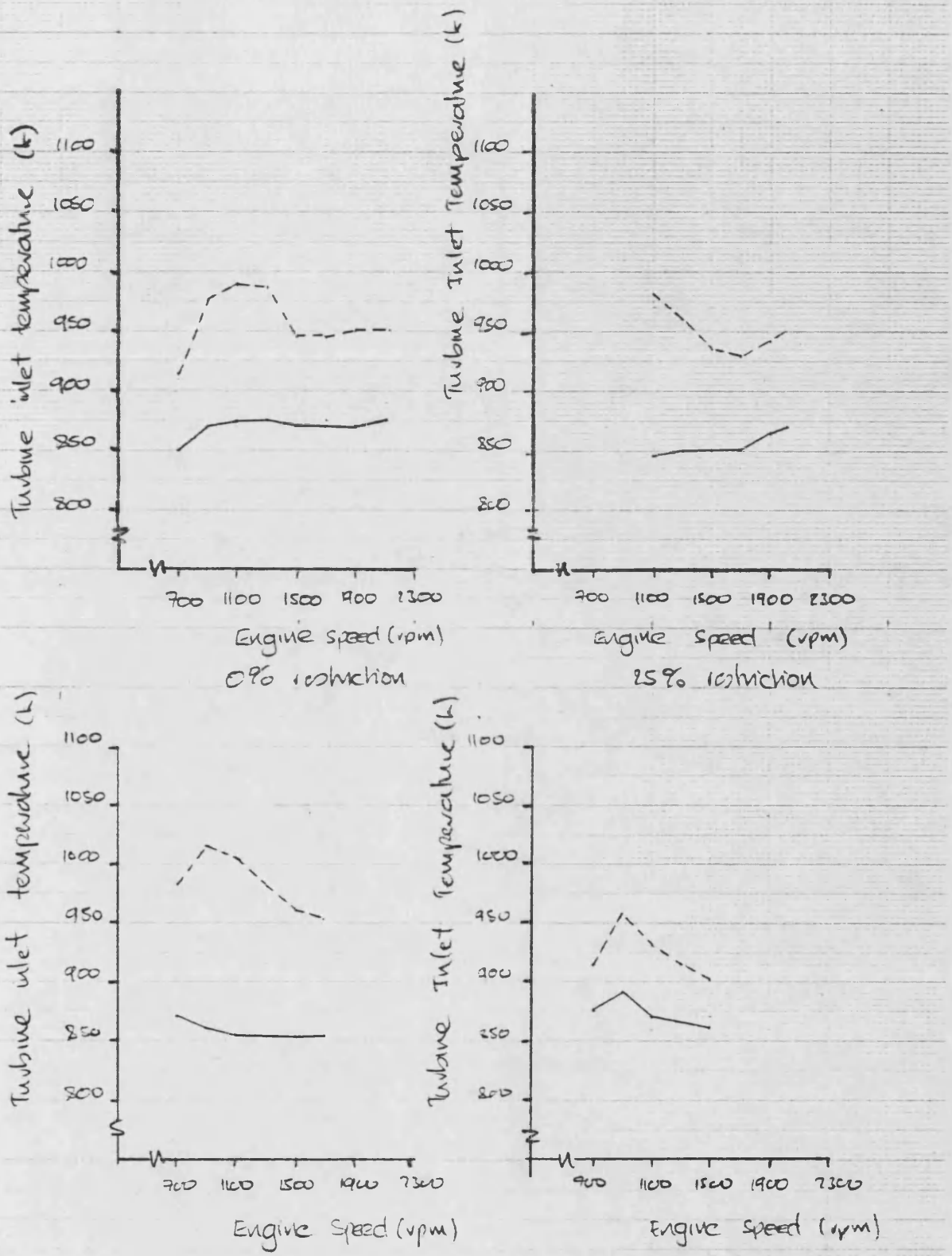


Figure 8.45 Turbine Inlet Temperature Results.

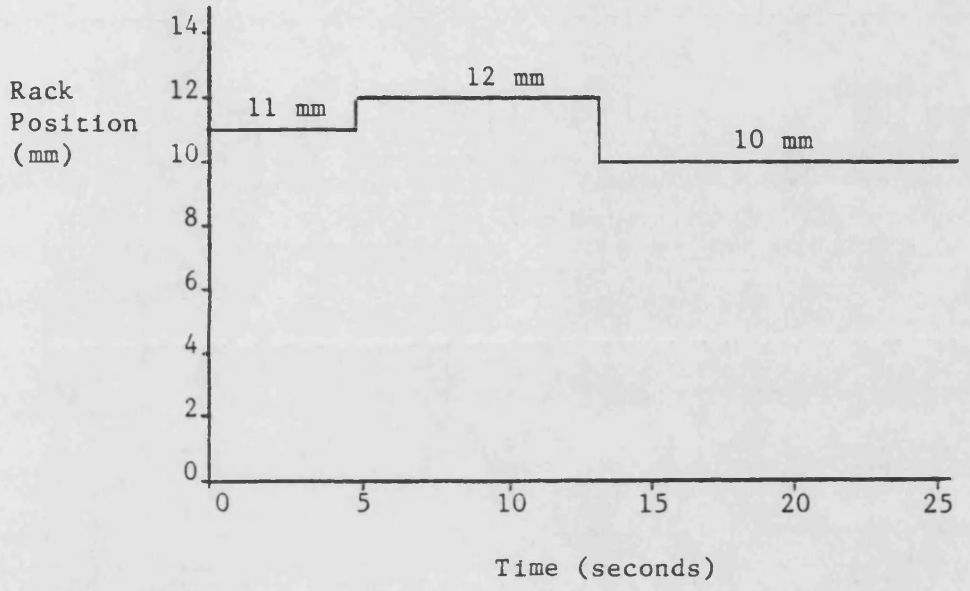


Figure 8.46a Fuel Rack Disturbance Signal.

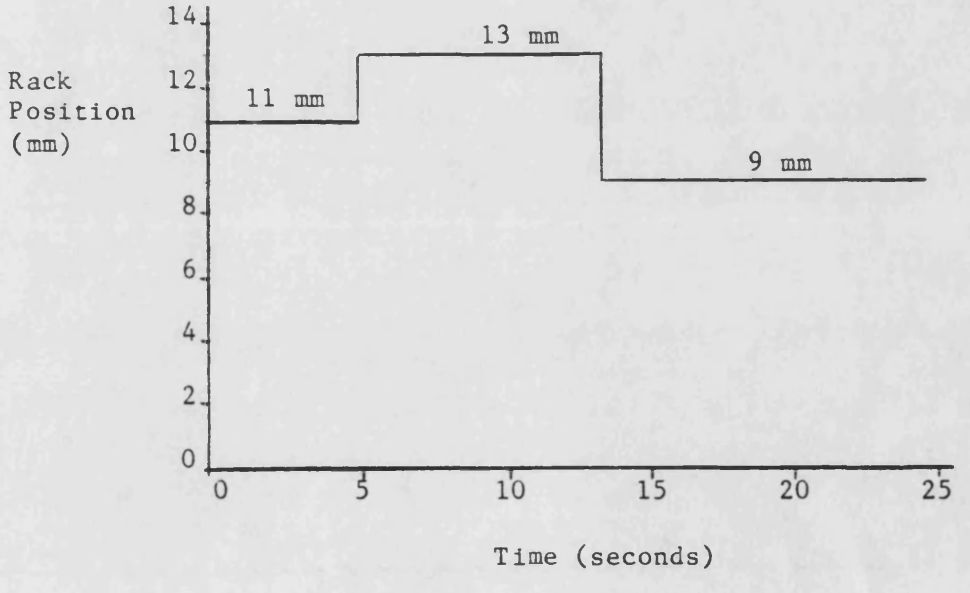


Figure 8.46b Fuel Rack Disturbance Signal.

Fig. 3.47 Rack response

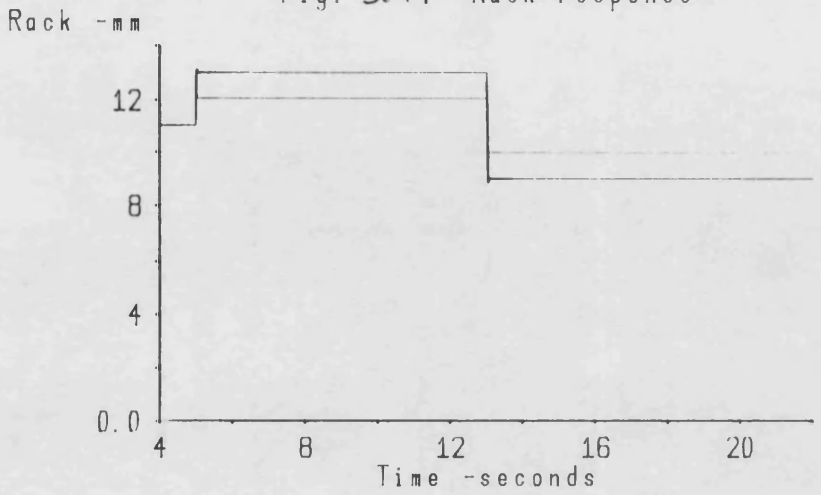


Fig. 3.48 Torque response

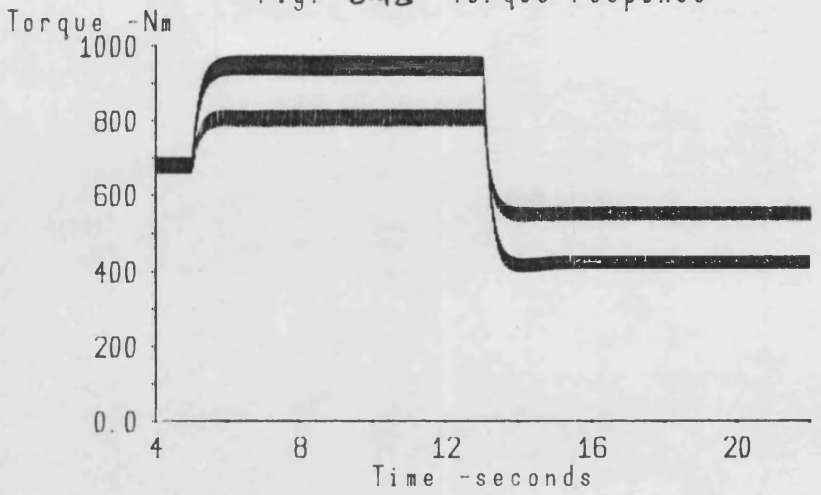


Fig. 3.49 Exhaust pressure response

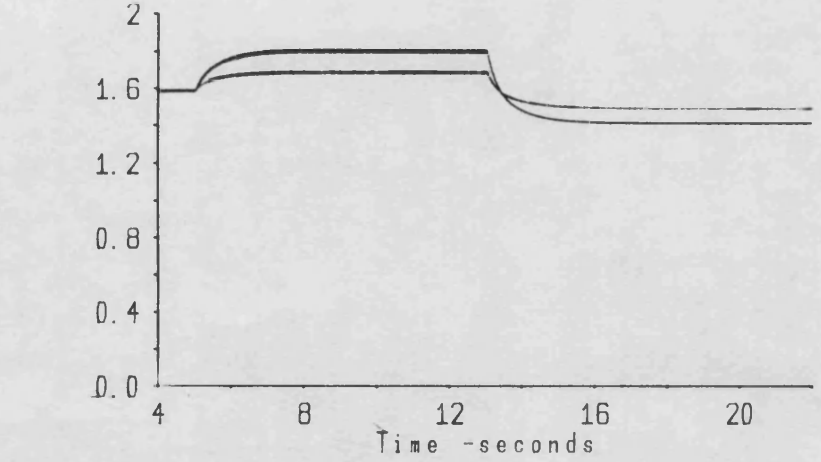


Fig. 3.50 Exhaust temperature response

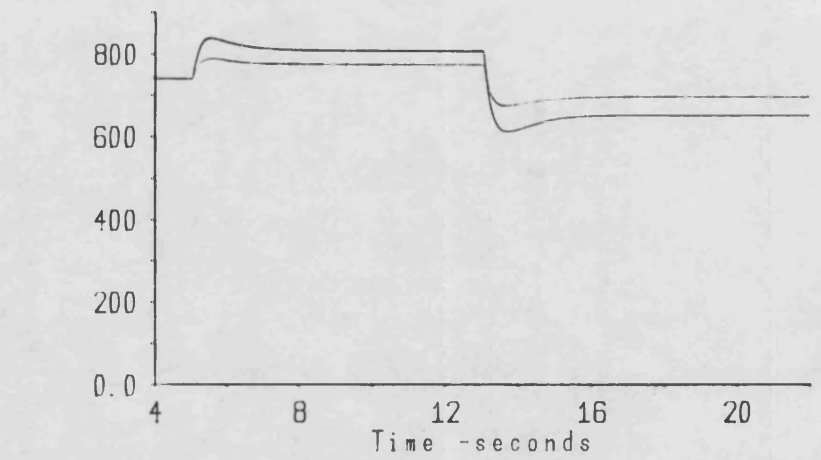


Fig. 8.51 Exhaust fuel-air response

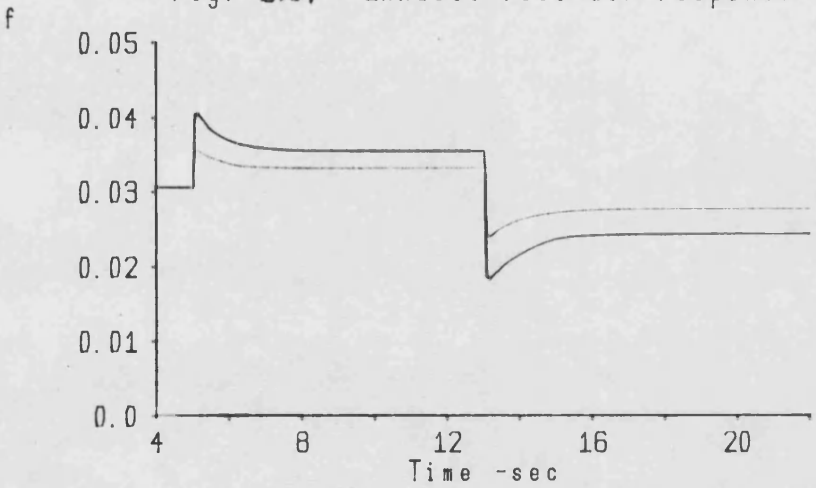


Fig. 8.52 Turbocharger speed response

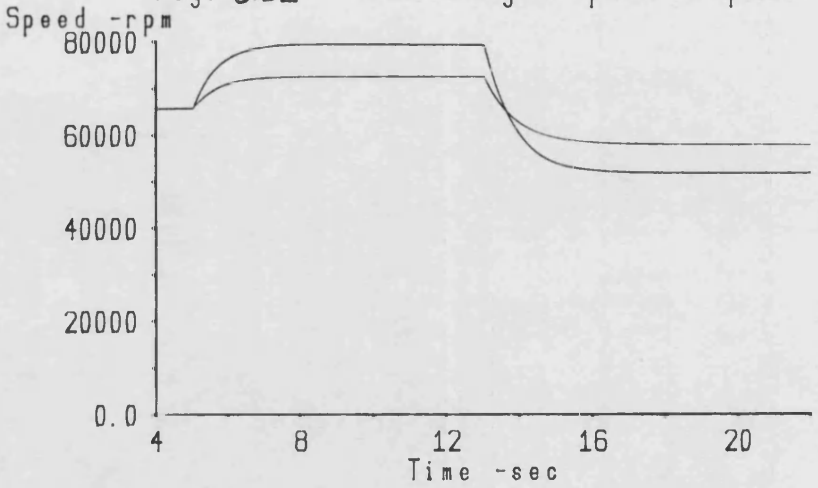


Fig. 8.53 Boost pressure response

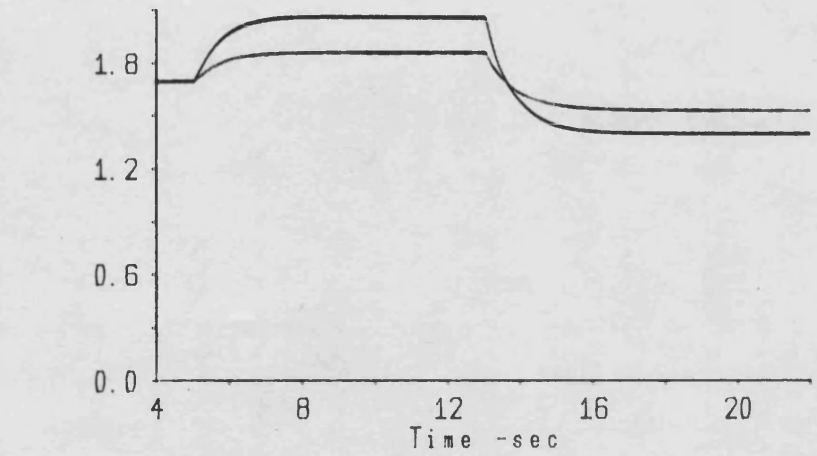
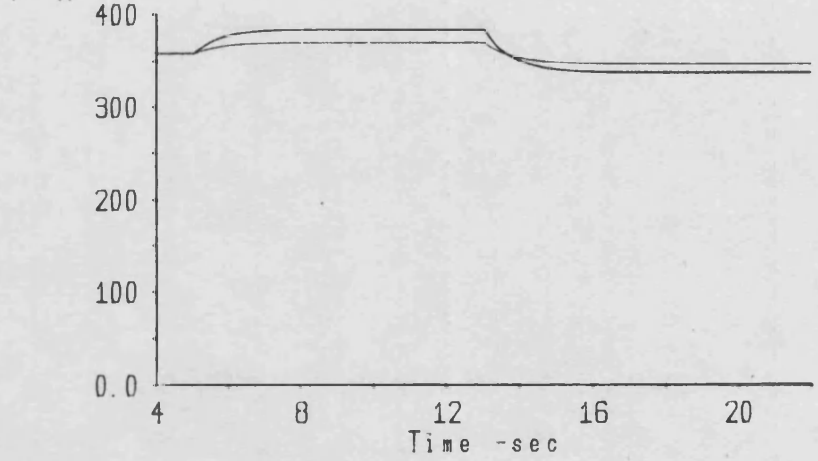


Fig. 8.54 Inlet temperature response



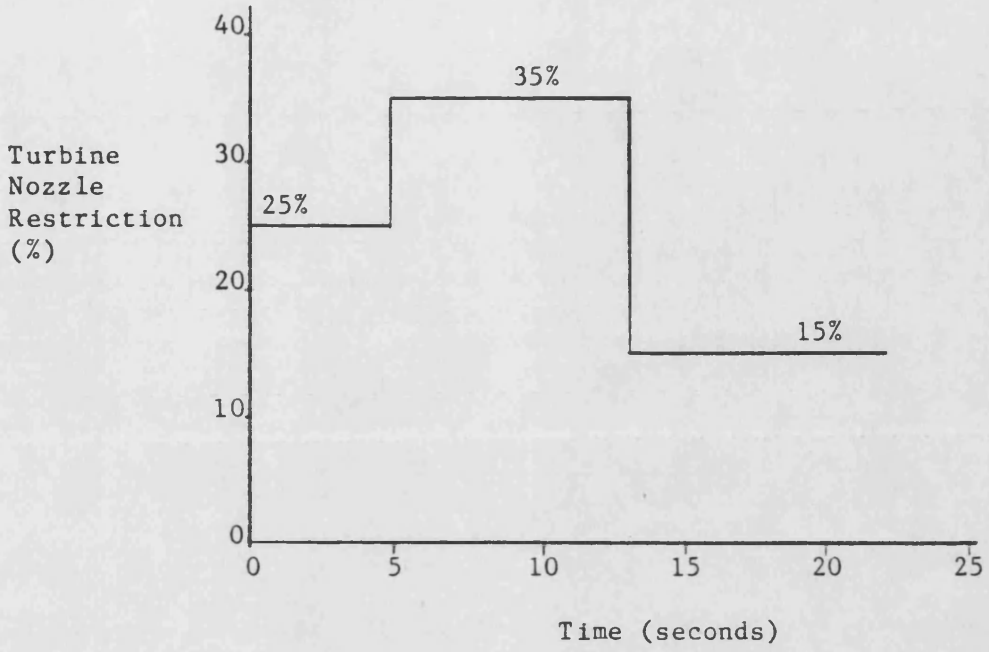


Figure 8.55a Turbine Nozzle Disturbance Signal.

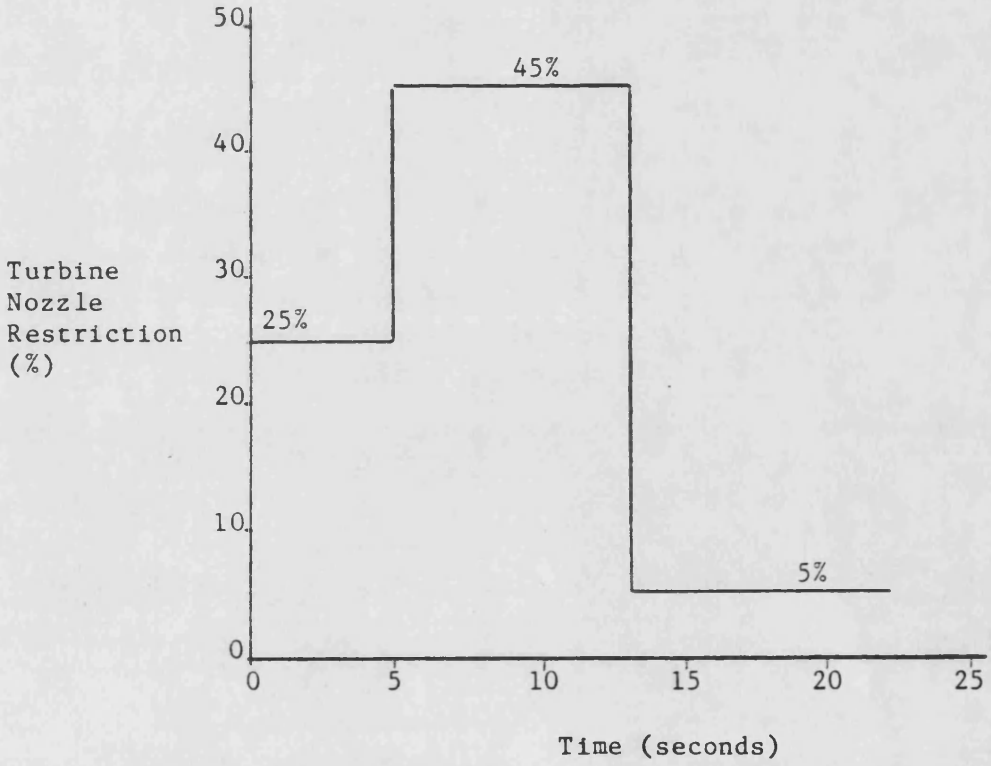


Figure 8.55b Turbine Nozzle Disturbance Signal.

Fig. 8.56 Turbine restriction

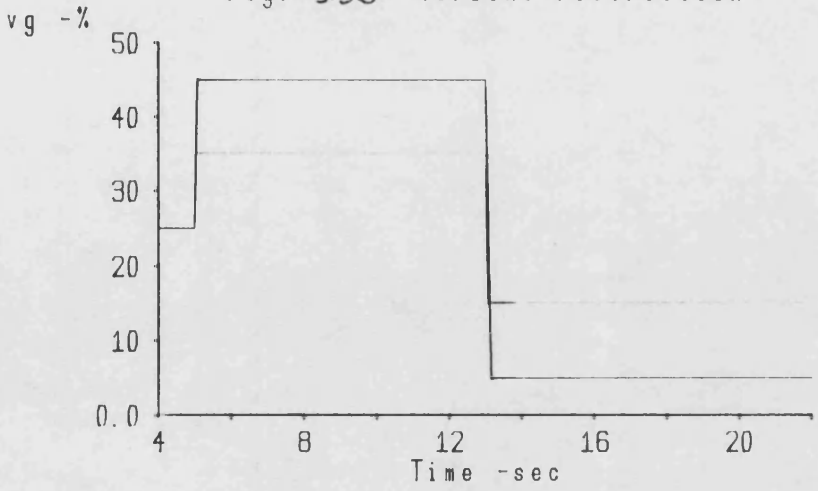


Fig. 8.57 Torque response -20%

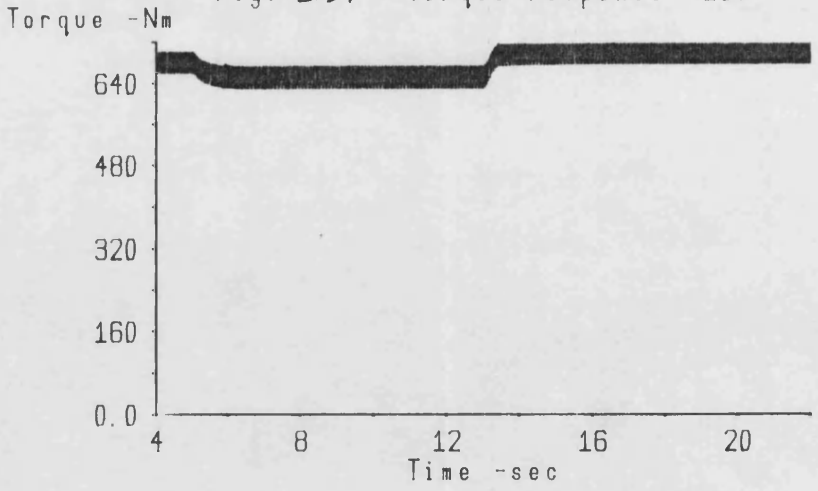


Fig. 8.58 Torque response -10%

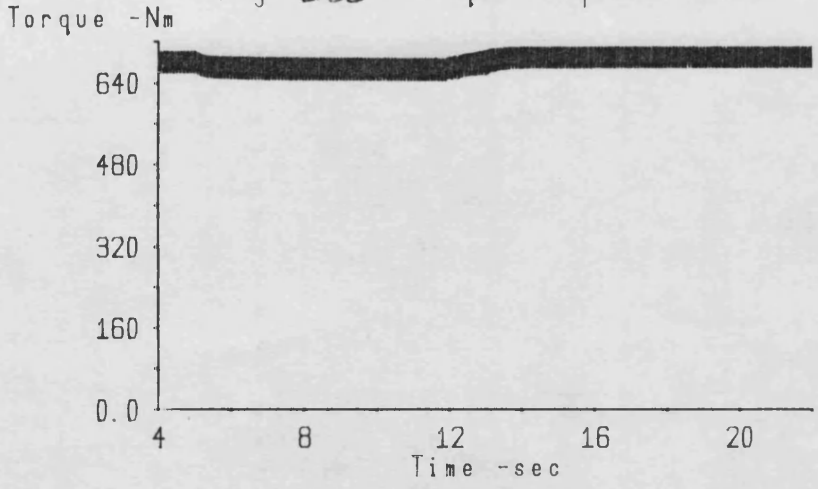


Fig. 8.59 Exhaust pressure response

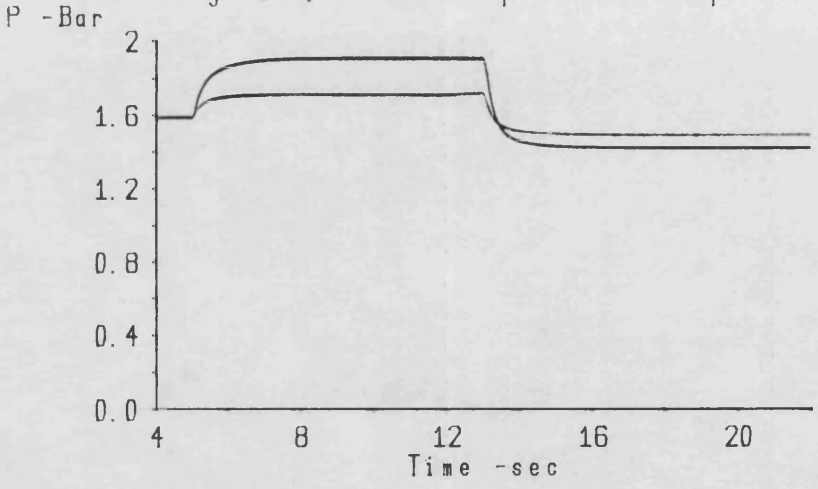


Fig. 8.60 Exhaust temperature response

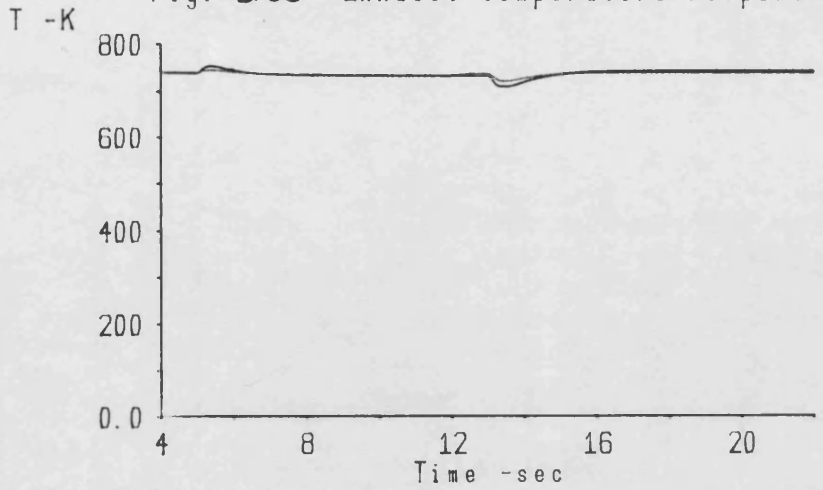


Fig. 8.61 Turbocharger speed response

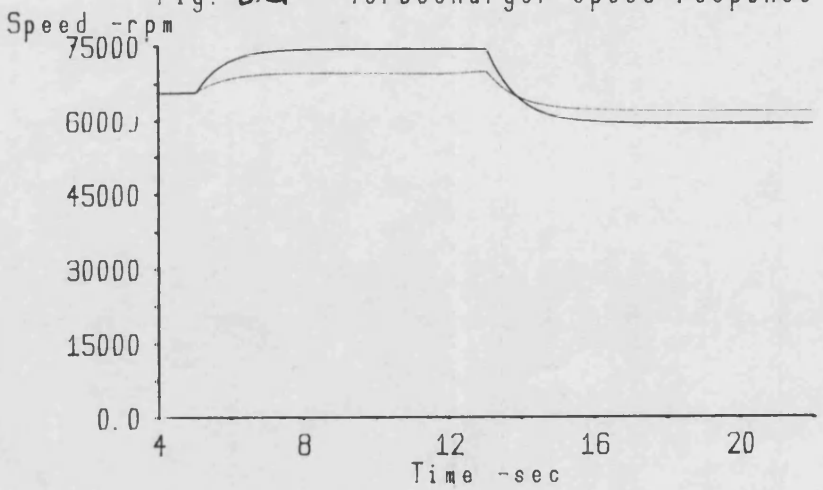


Fig. 8.62 Boost pressure response

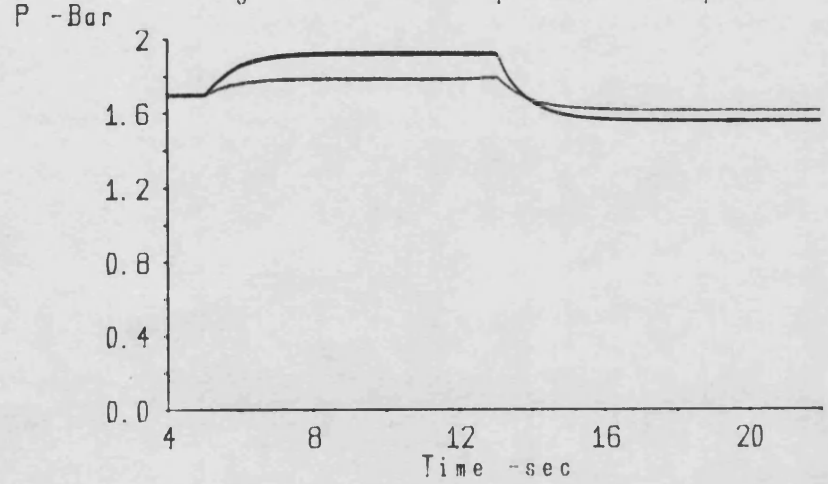
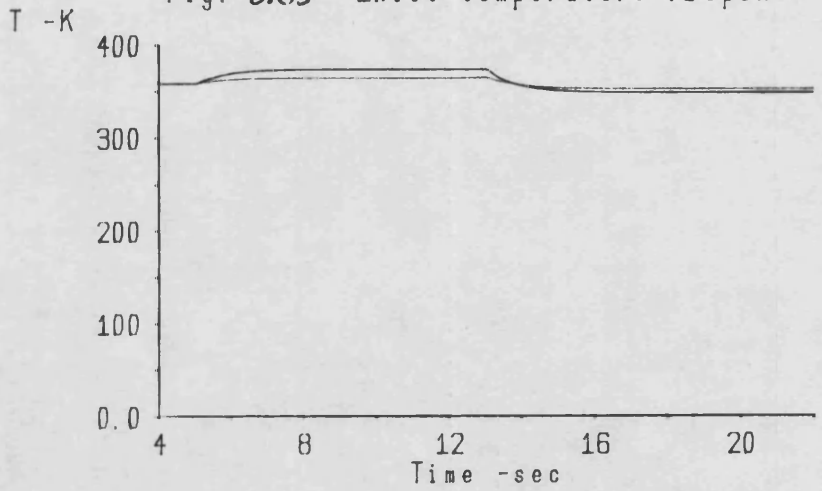


Fig. 8.63 Inlet temperature response



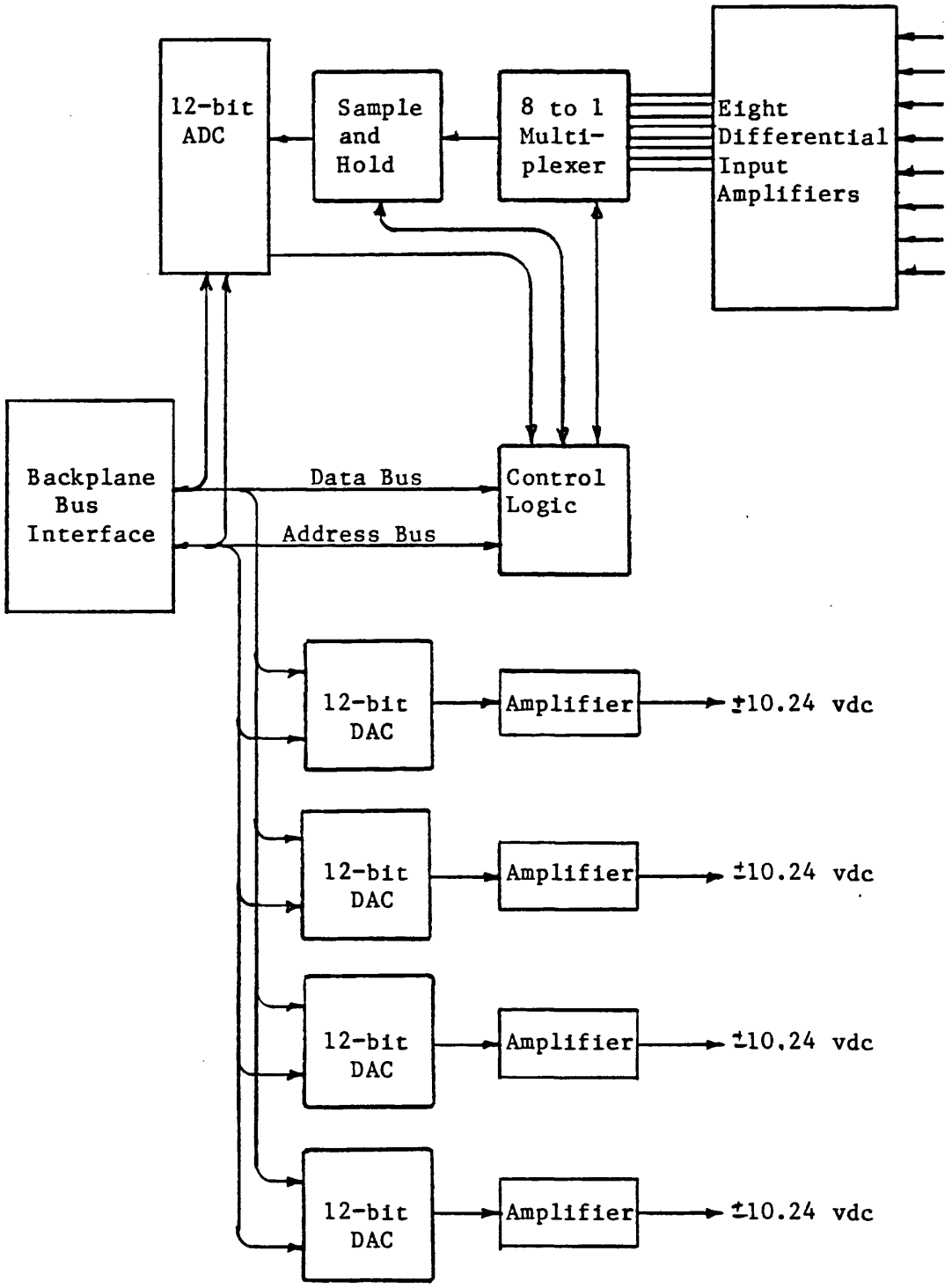


Figure 8.64 Schematic of Signal Conversion Card.

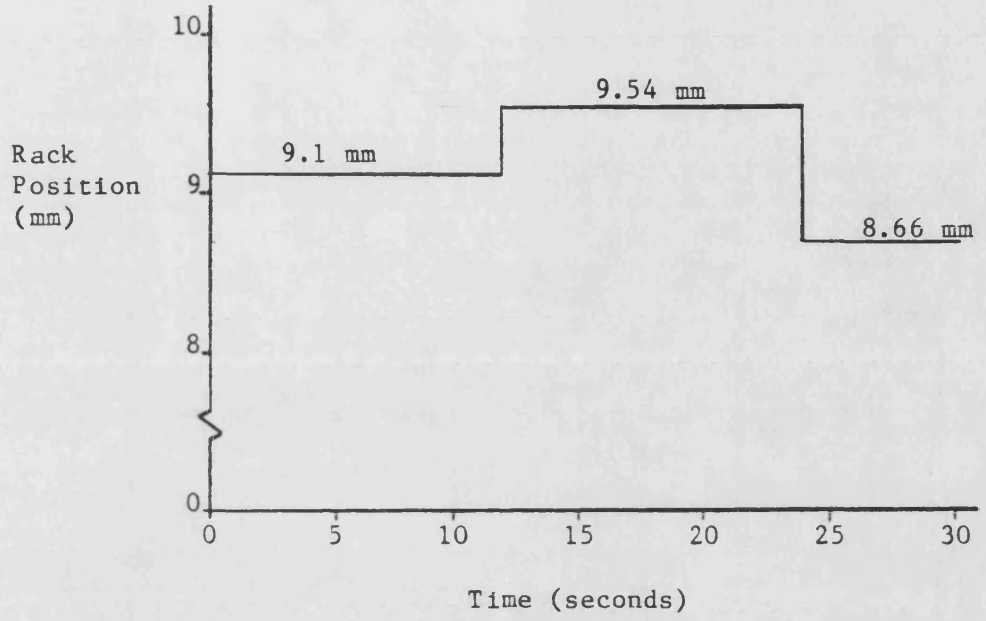


Figure 8.65 Fuel Rack Disturbance Signal.

Fig. 8.66 Fuel rack response

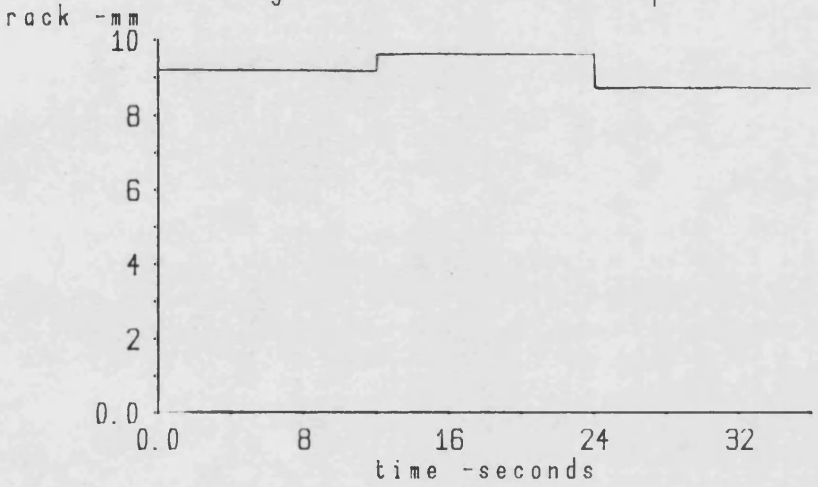


Fig. 8.67 Engine torque response

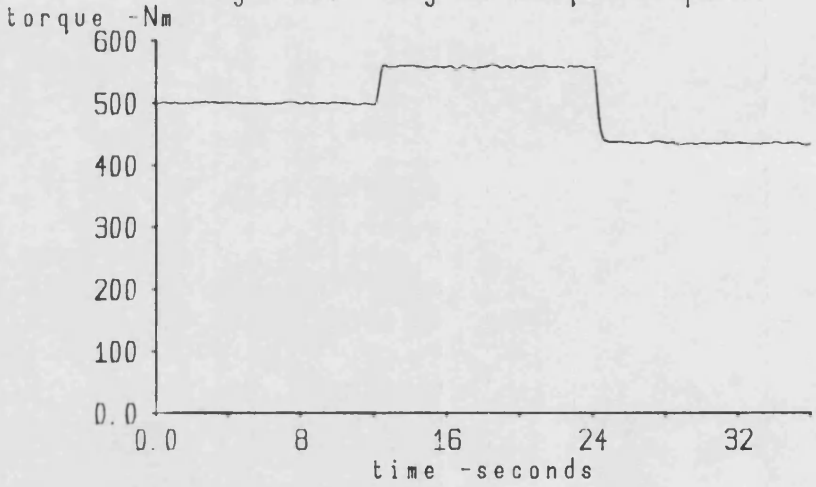


Fig. 8.68 Engine speed response

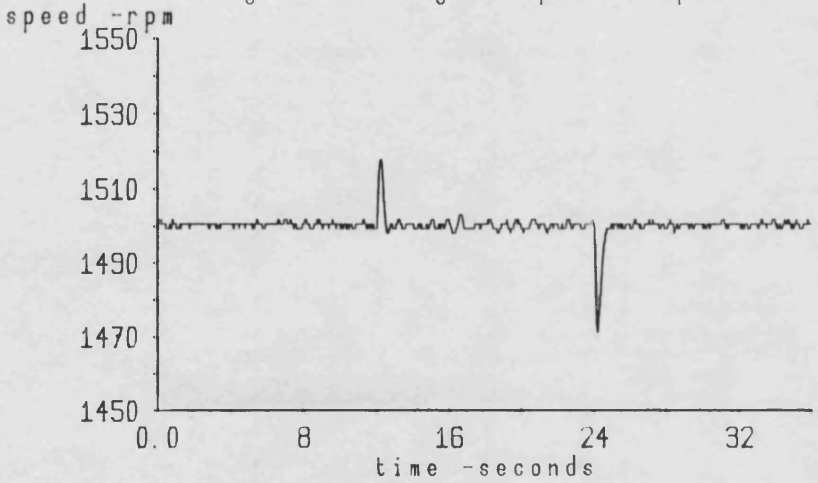


Fig. 8.69 Exhaust pressure response

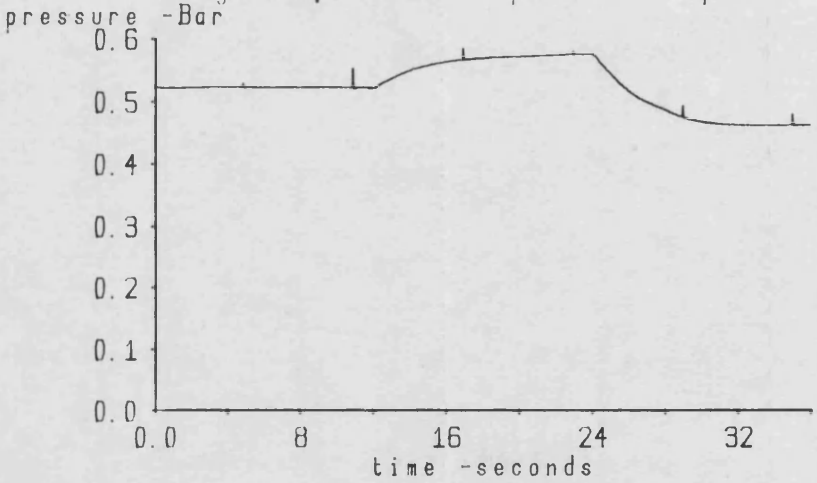


Fig. 8.70 Turbocharger speed response

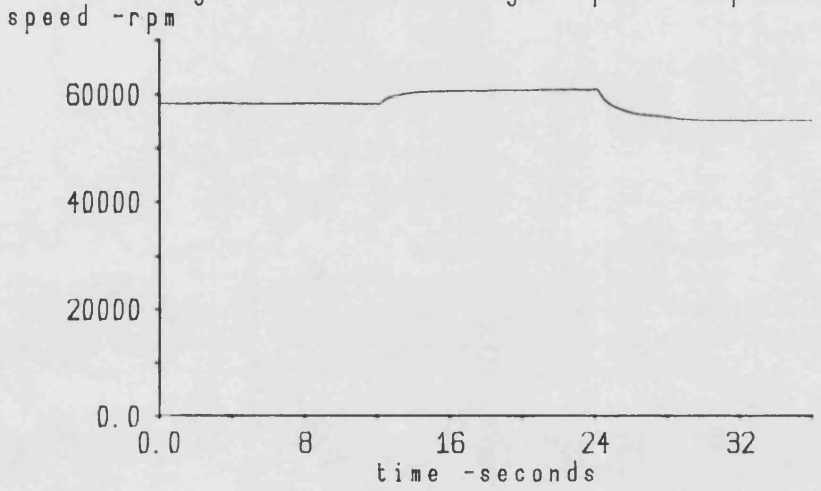
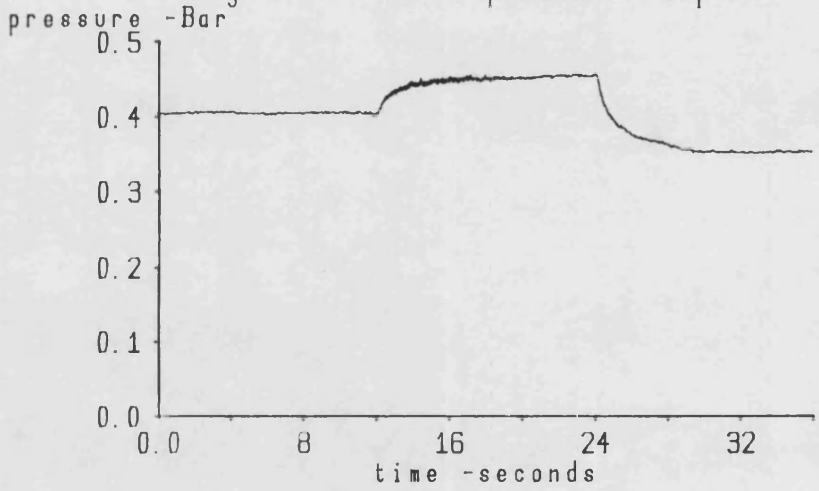


Fig. 8.71 Boost pressure response



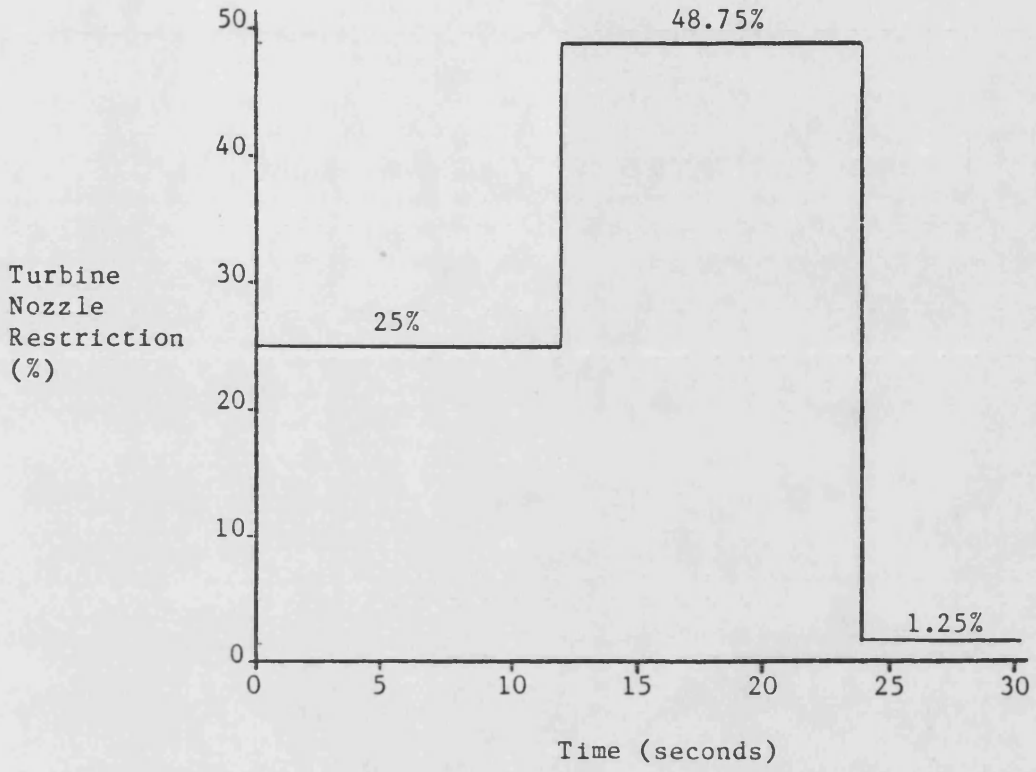


Figure 8.72 Turbine Nozzle Disturbance Signal.

Fig. 8.73 Turbine actuator response

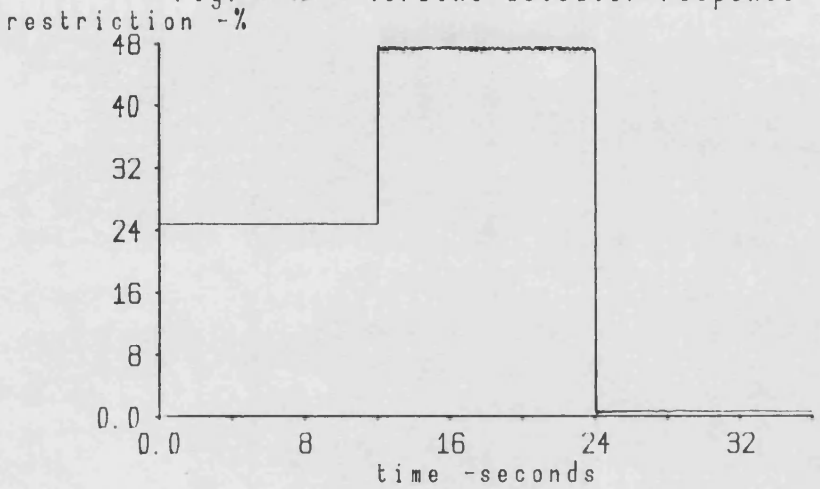


Fig. 8.74 Exhaust gas pressure reponse

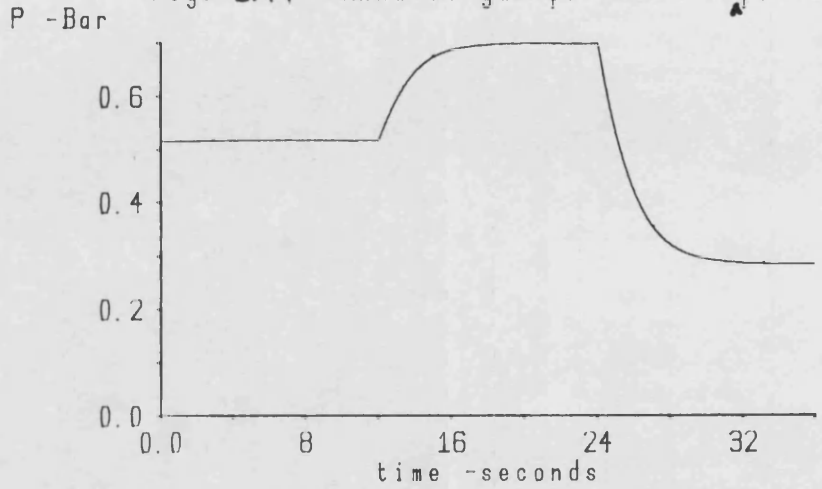


Fig. 8.75 Boost pressure response

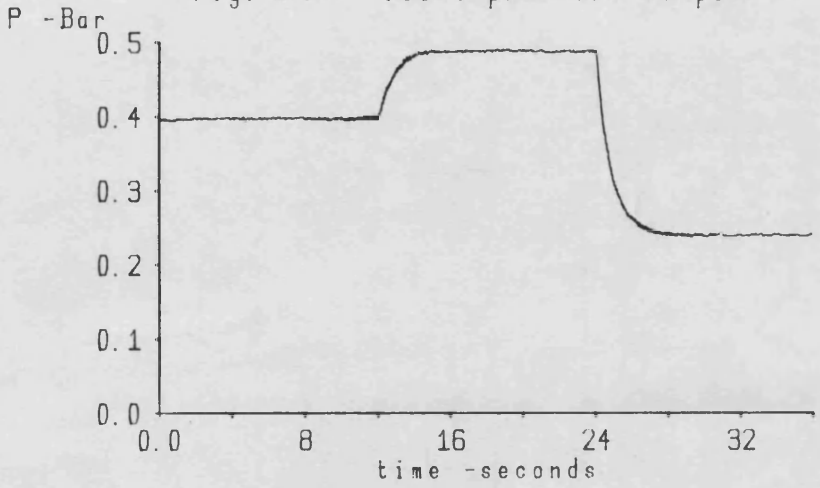
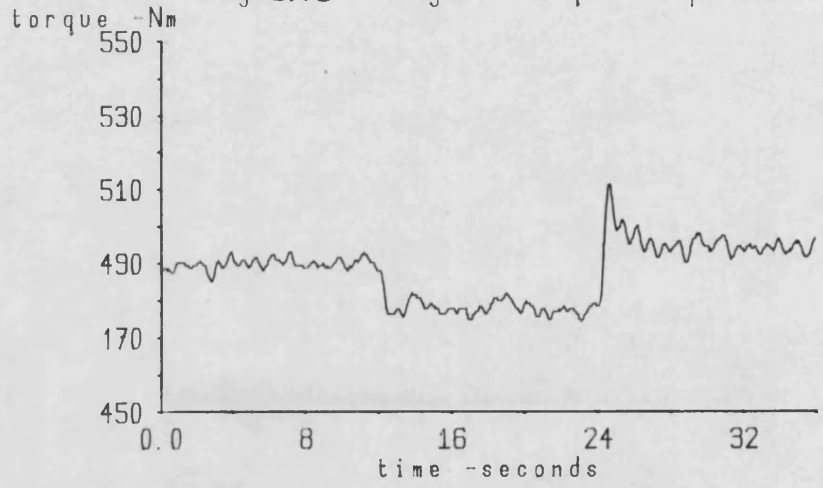


Fig 8.76 Engine torque response



CHAPTER 9

9.1 Results

By far the most important factor by which the success of the parallel solution of the engine model can be judged is the improvement in execution time compared with that obtained when computing the model serially (ie using a single processor).

Even using parallel processing, it is unlikely that a real time solution of the filling and emptying engine model will be achievable in the very near future. There is therefore, good reason to critically examine the simulation to identify those factors which have a major influence on run time, with a view to optimising their influence on speed. Some of these factors, such as integration step size and stability criteria, influence the speed irrespective of whether the model is computed in series, or in parallel. Other factors which influence speed are use of "look up" tables and, of course, the inherent speed of the processor itself.

This chapter presents results of experiments carried out to measure the speed improvement achieved by parallel processing, and the effect on speed of other factors, including those mentioned briefly above. The results are presented in four sections. The results presented in Section 9.2, show how the execution speed of the model depends upon the number of cylinders simulated and the number of processors used to perform the model calculations. In Section 9.3, results are presented which show how execution speed

depends upon simulated engine speed, the value used for the integration step size and the stability criteria. Results are also given which show the effect of using "look up" tables. Section 9.4 gives results showing how much faster the model is executed using the MC68020 based computer system, compared to the MC68000 based system. Finally, a comparison of the performance of the MC68000 and MC68020 based computer systems against conventional mini and mainframe computers which are typical of those used in much engine simulation work, is given in Section 9.5.

9.2 Effect of the Number of Cylinder Control Volume Models on Model Execution Speed

Whether, or not, a good speed improvement can be obtained using parallel processing depends very much upon the extent to which the calculations can be shared equally between the processors. This balance is particularly difficult to achieve for the engine model, because the calculations for the cylinder control volume models (which constitute a high proportion of the engine model calculations) vary by a considerable margin, depending upon which phase of the engine cycle is being calculated (see Chapter 5). The experiments described in this section were designed to assess this effect by measuring the execution times of models having different numbers of cylinders. To do this, a model was used which consisted solely of cylinder control volume models.

The experiments were all carried out with the engine operating at a steady speed of 1500 rpm, using an integration step size of 2 degrees and a solution tolerance of ±0.25%, these values being

typical of what would be used in practical simulation work. The test procedure was to start the model running at the appropriate operating condition, and after it had achieved a steady thermodynamic solution, to measure the time taken to compute the model response over a number of complete power cycles. The results are presented in two parts. The first set of results (Section 9.2.1) were obtained computing the model using an ideal task allocation, in which each processor is responsible for calculating one cylinder control volume model. The second set of results (Section 9.2.2) were obtained using a non-ideal task allocation in which each processor computes two cylinder control volume models.

9.2.1 Ideal Task Allocation Results

The execution times for one, two, three, four and five (‡) cylinder control volume models were measured, when computed in parallel using the ideal task allocation shown in Figure 9.1. The experiments were also repeated using a single processor (the serial condition). Details of the experiments are summarized in Table 9.1a, and Table 9.1b lists the times taken to compute the models over one power cycle, the speed up factors, and computational efficiencies achieved.

(‡) Because the computer system used only five slave processing nodes, the ideal task allocation could only be used for models having no more than five cylinders.

The execution times have been normalized with respect to the time taken to compute the single cylinder model, and are shown plotted in Figure 9.2a against the number of cylinders simulated. As expected, the results show that the execution time of the model increases as the number of cylinders simulated is increased, and that this occurs irrespective of whether the model is computed serially, or in parallel, - but not, of course, to the same extent. Figure 9.2a shows that when the model is computed serially, the increase in computational time for each additional cylinder simulated is greater than the execution time of the single cylinder model itself. This is because adding more cylinders to the simulation increases the occurrence of those periods of engine operation (such as scavenge), during which the required solution accuracy can only be achieved by repeated application of the corrector formula (5.2) by the integrator, and, in a case of exceptional difficulty, by a reduction in the integration step size. These iterations require recalculation of all the control volume models, not just the model(s) causing the stability problem. Consequently, as well as having to perform the additional calculations associated with the simulation of the additional cylinders, the existing cylinders require more stages to their calculation. When the model is computed serially, this results in an increase in the rate of change of execution time.

When computing the model in parallel, there is a second factor which increases the execution time of the model. Figure 9.2b shows how the execution time changes with the number of cylinders simulated, (when the model is computed in parallel) after allowing for the increase in the number of model calculations just shown to

occur. The figure shows an initial sharp rise in execution time, although the increase soon begins to decline when more than two cylinders are included in the engine model. The reason for the rapid increase in execution time for the two cylinder model is that the different computational requirements of the engine power phases have no effect on the execution time for the single cylinder model, but introduce significant delays in the computations for the two cylinder model. With the two cylinder model, one or other cylinder is always operating in the open phase of the engine cycle, which requires more calculation than the closed phase (corrector calculation stage), - because the valve flows have to be calculated. Consequently, the fastest rate at which the model corrector calculations can proceed is always determined by the speed at which the open phase calculations take place and this is the principle cause of the sudden increase in the execution time for the two cylinder model.

When simulating more than two cylinders, the execution time continues to increase (Figure 9.2b), but the rate of increase is less, since by this stage all the corrector calculations take place at a rate determined by the open phase requirements. This continued increase of execution time is due primarily to two factors. Firstly, the corrector calculations vary during the open phase: two valve flows are calculated during scavenge, whereas only one flow is computed during the induction and exhaust phases. Secondly, during the predictor calculations, the rate at which fuel burns is only evaluated during the combustion phase of engine operation. Both these factors delay the model calculations and hence increase execution time. When more cylinders are simulated (provided they

are at different angular positions on the engine crankshaft), a "maximum" value will be reached for the function shown in Figure 9.2b, when at every instant there is always one cylinder operating in scavenge, and another operating in combustion. If still more cylinders are simulated, the execution time of the model will continue to increase, but at a much reduced rate, which is caused solely by the delay introduced by inter-processor communication.

The improvement in execution speed obtained by computing the model in parallel can be seen from Figure 9.2a. The figure shows that the rate of change of execution time increases when computing the model serially, but decreases when computing the model in parallel. Figure 9.2c shows that their ratio (ie the speed up factor, s) increases almost linearly with the number of cylinders and can be represented quite accurately by the equation:-

$$s = 1 + 0.61.(c-1) \qquad (9.1)$$

where c is the number of cylinders. The computational efficiency is therefore given by:

$$\eta = \frac{1 + 0.61.(c-1)}{c} \qquad (9.2)$$

These equations have been used to estimate the speed up factor and computational efficiency which can be expected using an ideal task allocation, when computing 4, 6, 8, 10, 12 and 16 cylinders (which covers most engine configurations) and these estimates are listed in Table 9.2. They indicate that a major improvement in execution speed is possible, especially when simulating an engine having a

large number of cylinders.

9.2.2 Non-Ideal Task Allocation Results

Due to financial restraints, there were not sufficient processors available in the MC68000 based computer system to compute the TL11 engine model by allocating one processor to each computational task; consequently the engine model had to be computed using a non-ideal task allocation. The allocation was chosen with a view to "smoothing" the computational irregularities which occur between processors and hence improve computational efficiency.

The method chosen was to compute the two cylinders which are phased by a complete revolution of the engine on the same processor. Each processor then calculates at least one, but not more than two valve flows. This compares with the ideal task allocation where each processor has to calculate either no valve flow (during the closed periods of the engine cycle), one valve flow (induction and exhaust phases), or two valve flows (scavenge phase).

During these experiments, the execution time of a model consisting of 2, 4, 6, 8 or 10 cylinder control volume models was measured using the task allocations shown in Figure 9.3, and also serially using a single processor. Details of the experiments are summarized in Table 9.3a.

The execution time results, speed up factors and computational efficiencies measured during the experiments are listed in Table 9.3b and are shown plotted in Figures 9.4a to 9.4c. The execution time results are presented in terms of execution speed, ie

the number of revolutions completed by the model in one minute of actual time. The results were obtained operating the model using the same conditions as were used in the ideal task allocation experiments, presented in the previous section. Thus, for comparison, the ideal task allocation results are also shown plotted on these figures.

As expected, the results show that when the model is computed using the non-ideal task allocation, the execution speed of the model is slightly faster than half the execution speed of the same model when computed using the ideal task allocation (Figure 9.4a). However, half the number of processors perform the calculations and consequently computational efficiency is improved, - typically by some 10 to 20% depending upon the number of cylinders, as is shown in Figure 9.4c. The figure also shows that computational efficiency declines with the number of cylinders. As was the case when computing the model using the ideal task allocation, the decline is due primarily to the different computational requirements of the cylinder models during the different phases of engine operation. However, because the calculations to be performed by each processor are more uniform when using the non-ideal task allocation, the rate of decline is less.

The results given in this section show that by using each processor to compute two cylinder control volume models, an improved balance is achieved in the calculations to be performed by the processors, and hence the speed up factor and computational efficiency are improved. However, the improvement is only achieved at the expense of a very significant reduction in execution speed

(Figure 9.4a) and whichever scheme may be adopted, it must be a compromise between execution speed and computer system cost.

9.3 TL11 Engine Model Experiments

The results given in the previous section of this chapter were obtained using a model consisting solely of cylinders. The addition of manifolds to the model increases the computational difficulties of the model and inevitably increases execution time. When flow takes place between a manifold and cylinder, the gas pressure in the manifold tends towards the pressure in the cylinder and the pressure differential becomes small, - especially when the rate of change of cylinder volume is small. To prevent the calculated mass flow spuriously oscillating between the two volumes, the integrator has to apply its corrector formula (5.2) very frequently and use a small step size, which obviously increases execution time.

It was decided to make measurements using the TL11 engine model described in Chapter 4, and results are presented in this section showing how fast the model is computed, the improvement in execution speed achieved by computing the model in parallel, the computational efficiency, and how the execution speed compares with a real time solution. Section 9.3.1 presents results for the "standard" version of the TL11 engine model and Section 9.3.2 for a version of the model which uses "look up" tables.

9.3.1 TL11 Engine Model Experiments - No Look Up Tables

The most important purpose of these experiments was to measure

the improvement in the execution speed of the TL11 engine model, achieved by computing the model in parallel. To show how the improvement depends upon the number of processors, the execution time was measured when using one, two, three, four and five processors. These results are given in Section 9.3.1.1. Other experiments were carried out to show the effect of integration step size and stability criteria on execution time and these results are given in Section 9.3.1.2.

9.3.1.1 The Speed Benefit of Parallel Processing

Results are presented in this section which show how execution speed changes as the number of processors used to compute the engine model is increased from one to five. The task allocations used in the experiments are shown in Figure 9.5. The figure shows that in some experiments, the maximum number of control volume models computed by a processor was the same as when fewer processors were available (eg when computing the model using three and four processors). It is shown below that this gives rise to definite breakpoints in the execution speed improvement relationship.

A summary of the experiments performed is given in Table 9.4a and the execution speeds recorded are listed in Table 9.4b. Table 9.4b also lists the improvements in execution speed and computational efficiencies achieved.

As expected, the results show that the execution speed of the model improves as more processors are used, reaching a speed improvement of 3.3 times when using five processors, - which is a

significant improvement. The execution speed results are shown plotted in Figure 9.6; this shows how the speed improvement is obtained, and is of considerable practical importance when deciding how many processors to use when computing a model.

Figure 9.6 shows that only a very small improvement in the execution speed of the model is achieved by using four processors, compared with using three. The reason for this is that in both cases at least one processor has to compute three control volume models (see Figure 9.5) and consequently the execution speeds are similar. For this reason, it is to be expected that little improvement in execution speed will be achieved by increasing the number of processors to six, seven or eight compared with using five processors, since in all these cases at least one processor is responsible for computing two control volume models. However, when nine processors are used a significant improvement in execution speed should occur because each processor is then responsible for computing only one control volume model. Clearly, it is important to recognise the existence of these breakpoints when planning the installation of a parallel computer system.

9.3.1.2 Sensitivity of Execution Speed to Integration Step Size and Stability Criteria

These experiments were carried out to measure how integration step size and stability criteria affect the execution speed of the engine model when it is computed in parallel. During the experiments the execution speed of the model was measured using integration step sizes of 1, 2 and 4 degrees, and stability criteria

of ±0.1%, ±0.25% and ±0.5%. These values were chosen primarily to assess their effect on model execution speed, although they also cover values which are likely to be chosen when using the model to predict engine performance.

The experiments performed are summarized in Table 9.5a and the execution speed results are listed in Table 9.5b. As expected, the results show that, in general, increasing integration step size improves execution speed. However, if an integration step size is used which results in the model often being unable to achieve the required solution accuracy, then the execution time will be longer than if a smaller step size had been used. The results also show that the execution speed of the model gradually improves as the integrator tolerance is relaxed. This is also expected, because when operating with a relaxed stability criteria, the integrator is able to achieve the required solution accuracy with fewer applications of its corrector formula. A limiting execution speed (the fastest) will be reached when all the control volume calculations achieve a successful result on the first application of the corrector formula.

The results show that a poor choice of integration step size makes the execution time of the model unduly long. If the step size chosen is unnecessarily small, the accuracy of model solution is higher than required, and the execution time is made longer. Conversely, if the step size chosen is too large, the corrector formula has to be applied many times, and the step size reduced in order to achieve the required accuracy, and, once again, the execution time is made longer. Clearly, in order to achieve the

shortest execution time, the correct choice of integration step size is important.

To measure how sensitive the "optimum step size" is to simulated engine speed, the execution speed of the TL11 model was recorded when operating at speeds of 750 to 2250 rpm in increments of 250 rpm. At each engine speed, the execution speed of the model was recorded using integration step sizes of 1, 1.5 and 2 degrees. An integrator stability criteria of ±0.25% was used in all the experiments. A summary of the experiments is given in Table 9.6a and the execution speeds recorded are listed in Table 9.6b. Table 9.6b also lists the ratio of model execution speed to engine speed (referred to as "real time factor") and for the experiments performed using the 2 degree step size, the speed up factor and the computational efficiency are also shown. The execution speed of the model is shown plotted in Figure 9.7a and the "real time factor" in Figure 9.7b.

The first observation to be made is that the speed up factor (Table 9.6b) and hence the computational efficiency are almost independent of the simulated engine speed; the speed improvement achieved (of the order 3.3) corresponds to a computational efficiency of about 66%.

As predicted, the execution speed of the model (Figure 9.7a) improves as simulated engine speed increases, until an engine speed is reached at which the model can achieve the required accuracy of solution with a single application of the corrector formula (5.2). Beyond this engine speed, the execution speed of the model is constant.

The effect of making a poor choice of integration step size on the execution speed of the engine model is clearly demonstrated by Figure 9.7a. For example, with an engine speed of 750 rpm the execution speed of the model is 1.6 times faster when using an integration step size of 1 degree, than it is when using a step size of 2 degrees. Conversely, when the engine speed is 2250 rpm, the execution speed of the model is 1.56 times faster with a 2 degree step size, than it is with a 1 degree step size. When the simulated engine speed is low (less than 1100 rpm) the fastest model run time occurs using the smallest of the integration step sizes, that of one degree. This is because at low engine speed the model has great difficulty in achieving the required solution accuracy. Consequently, when using the larger integration step sizes, it has to apply the corrector equation (5.2) more times, and in a case of exceptional difficulty it has to reduce the step size. As the simulated engine speed is increased (from 750 rpm), the required solution accuracy becomes easier to achieve and at an engine speed of 1100 rpm, the execution speeds achieved using the 1.0 and 1.5 degree step sizes become identical. Between an engine speed of 1100 and 1250 rpm the model is computed fastest using a step size of 1.5 degrees, and at higher speeds, using the 2 degree step size. At high engine speed, the execution speed of the model is fastest when using the large integration step size because the model then experiences little difficulty in achieving the required accuracy. Clearly, the effect of step size on execution speed is significant; consequently in a dynamic diesel engine model, there is a very strong case for making step size a function of engine speed, in order to ensure that the highest execution speed is always achieved.

Figure 9.7b shows that the real time factor changes considerably with engine speed for each integration step size. These results also strongly suggest that real time solution of the model will be first achieved at low engine speeds. This is particularly encouraging, since a prime application of this work is seen to be for the control and condition monitoring of slow speed diesel engines typical of those used in maritime applications where economy of operation is vital.

9.3.2 TL11 Engine Model Experiments - Using Look Up Tables

One way in which a further improvement in the execution speed of the model can be obtained is to use look up tables to store, and subsequently retrieve, the results of computationally expensive calculations (see Chapter 5). To assess the improvement in execution speed, it was decided to repeat one of the experiments described in the previous section using a version of the TL11 engine model which used look up tables. Details of the experiment are summarized in Table 9.7a.

The execution speeds recorded during the experiments are listed on Table 9.7b. For comparison, the table also lists the execution speed which was obtained using the standard version of the engine model. The results show that the use of look up tables resulted in a considerable improvement in the execution speed of the model, of the order 1.6 times faster, over the entire speed range of the engine.

The accuracy attainable using look up tables obviously depends

upon the increment size used for the independent variable(s) in the table entries; - a high accuracy requiring large memory arrays. However, with the falling cost of memory devices and the dramatic increase in their storage capacity, the use of look up tables to obtain the improvement in execution speed is becoming much more attractive.

9.4 Results Obtained Using the MC68020 Computer System

The experiments described in this section were carried out to measure how fast the engine model can be computed using the MC68020 based computer system which was described in Chapter 6. The system employs three MC68020 based slave processing nodes, hence when it was used to compute the TL11 engine model, each processor was responsible for computing three control volume models. The task allocation used in the experiments is shown in Figure 9.5(iii) and the experiments performed are summarized in Table 9.8a.

During the experiments, the execution time of the model was measured when computed in parallel (using three processors) and also when computed using a single processor. The experiments were also carried out with, and without, the use of the MC68881 floating point co-processor. This was done in order to assess what proportion of the total speed improvement achieved, can be attributed to the MC68020 processor and what is due to the MC68881 co-processor. When the model is computed without the use of the co-processor, the MC68020 performs floating point operations using software.

The execution speeds of the model recorded during the experiments are listed on Table 9.8b, together with the "parallel computing speed up factors" and computational efficiencies. The table also lists the results obtained using the MC68000 based system when performing the same experiment.

The results show that a very considerable improvement in the execution speed is achieved using the MC68020 based system; indeed, the model is computed nearly three times faster even without the co-processor. When the co-processor is used a further significant improvement in execution speed occurs, (approximately doubling the speed again), which incidentally, clearly demonstrates how "floating point" intensive are the engine model calculations. Overall, the model is computed some 6 - 7 times faster using the MC68020 processor and MC68881 co-processor than it is when using the MC68000 based system.

9.5 Comparison of the MC68000 and MC68020 Based Computer Systems Against Conventional Mini and Mainframe Systems

This section gives a comparison of the speed performance of the MC68000 and MC68020 based computer systems compared to conventional mini and mainframe computer systems of the type commonly used in engine simulation work. The execution speed measurements on the mini and mainframe computer systems were made using the engine simulation program SPICE [9.1] (Simulation Program for Internal Combustion Engines), which is written in FORTRAN 77 [9.2] and is based on a filling and emptying model very similar to that used in this work.

The experiments simulated a truck sized diesel engine having the same configuration as the TL11 engine model, ie six cylinders, one inlet and two exhaust manifolds and turbocharged. The model (SPICE) was run on a VAX 11/750 mini computer, VAX 11/785 super mini computer and an ICL 3980 mainframe computer, and the execution times were measured. The experiments were also performed (with the TL11 engine model) using the MC68000 and MC68020 based parallel computer systems, for which non-ideal task allocations had to be used as shown in Figure 9.5(iii) for the MC68020 system and 9.5(v) for the MC68000 system. Details of the experiments are summarized in Table 9.9a and the execution speeds measured are given in Table 9.9b; Table 9.9b also lists the "real time" factors.

The results show that the performance of the MC68000 and especially the MC68020 based computer system compares extremely favourably with that of the mini and mainframe computers tested. In fact, the MC68020 based system, computes the engine model faster than any of the other systems, even when each slave processor has to compute three control volume models. Clearly, if a relatively few more processors had been available to permit the use of an ideal task allocation, then the speed advantage of the MC68020 system would have been even greater, as the estimated execution times given on Table 9.9b show.

9.6 Summary

This chapter has given results showing how the execution speed of a filling and emptying engine model is improved by computing the model in parallel. The speed improvement achieved was less than the

maximum speed improvement possible (equal to the number of processors), primarily because it was not possible to obtain an exact balance in the calculations being performed by the individual processors. Nevertheless, a considerable improvement in execution speed has been demonstrated, (for example an improvement of the order 3.3 when computing the TL11 engine model using five processors) and a significantly larger speed improvement could have been achieved, if a relatively few more processors had been available.

Results also show that a further improvement in execution speed can be achieved by using "look up" tables, which when computing the model using the MC68000 based system is of the order of 1.6.

Various experiments have been carried out to measure the sensitivity of execution speed to integration step size and stability criteria. The results show that a considerable execution time penalty results when using inappropriate values, and that the step size which results in the fastest computation, changes quite significantly with simulated engine speed.

Finally, results are given which show that the execution speed performance of the parallel computer system, compares extremely favourably with the performance obtained using conventional mini and mainframe computer systems, and of course it also has a significant size and cost advantage over them.

9.7 References

9.1 S Charlton.
SPICE, Simulation Program for IC Engines (User Manual).
April 1986, School of Mechanical Engineering, University of Bath.

9.2 VAXVMS Fortran Programmers Manual.
Sept 1984, Digital Equipment AA-Z212A-TE.

o Configurations Simulated

All simulations were carried out with constant manifold conditions

Number of Cylinders	Cylinder Crankshaft Position (degrees)
1	0
2	0, 360
3	0, 240, 480
4	0, 180, 360, 540
5	0, 120, 240, 360, 480

o Engine Operating Condition

engine speed 1500 rpm
fueling 0.06 grams of fuel per injection
static fuel injection timing 22 degrees btdc

o Integrator Tolerance and Step Size

integrator tolerance ±0.25%
integrator step size 2 degrees

Table 9.1a Cylinder Control Volume Models - Summary of Ideal Task Allocation Experiment.

Number of Cylinders	Serial Processing	Parallel Processing			
	Time per Cycle (secs)	Time per Cycle (secs)	Speed up Factor	Number of Slaves	Computational Efficiency (%)
1	18.2	18.2	1.0	1	100.0
2	38.3	24.3	1.57	2	78.8
3	62.1	28.0	2.22	3	73.9
4	87.3	30.9	2.83	4	70.6
5	115.4	33.6	3.43	5	68.7

Table 9.1b Cylinder Control Volume Models - Ideal Task Allocation, Execution Time Results.

Number of Cylinders Simulated	Number of Processors	Speed Up Factor	Computational Efficiency (%)
4	4	2.8	71
6	6	4.0	67
8	8	5.3	66
10	10	6.5	65
12	12	7.7	64
16	16	10.1	63

Table 9.2 Cylinder Control Volume Models - Ideal Task Allocation, Predicted Speed Improvement.

o Configurations Simulated

All simulations were carried out with constant manifold conditions

Number of Cylinders	Cylinder Crankshaft Position (degrees)
2	0, 360
4	0, 180, 360,540
6	0, 120, 240, 360, 480, 600
8	0, 90, 180, 270, 360, 450, 540, 630
10	0, 72, 144, 216, 288, 360, 432, 504, 576, 648

o Engine Operating Condition

engine speed 1500 rpm
fueling 0.06 grams of fuel per injection
static fuel injection timing 22 degrees btdc

o Integrator Tolerance and Step Size

integrator tolerance $\pm$0.25%
integrator step size 2 degrees

Table 9.3a Cylinder Control Volume Models - Summary of Non-Ideal Task Allocation Experiment.

Number of Cylinders	Serial Processing	Parallel Processing			
	Execution Speed (rpm)	Execution Speed (rpm)	Speed up Factor	Number of Slaves	Computational Efficiency (%)
2	3.15	3.15	1.0	1	100.0
4	1.38	2.4	1.74	2	87.0
6	0.827	2.0	2.41	3	80.5
8	0.565	1.74	3.07	4	76.9
10	0.45	1.73	3.84	5	76.8

Table 9.3b Cylinder Control Volume Model - Non Ideal Task Allocation Execution Time Results.

o Engine Configuration

6 cylinder turbocharged TL11 engine with one inlet manifold and two exhaust manifolds

Number of Cylinders	Cylinder Crankshaft Position (degrees)
6	0, 120, 240, 360, 480, 600

o Engine Operating Conditions

engine speed 1500 rpm
fueling 0.06 grams of fuel per injection
static fuel injection timing 22 degrees btdc
turbine nozzle fully open
"infinite" inertia load

o Integrator Tolerance and Step Size

integrator tolerance ±0.25%
integrator step size 2 degrees

Table 9.4a TL11 Engine Model - Summary of Experiment.

Number of Processors	Execution Speed (rpm)	Speed Up Factor	Computational Efficiency (%)
1	0.434	1.0	100.0
2	0.69	1.59	79.5
3	1.04	2.4	80.1
4	1.09	2.53	63.2
5	1.45	3.33	66.6

Table 9.4b Execution Time Results for TL11 Engine Model when Computed using 1 to 5 Processors.

o Engine Configuration

6 cylinder turbocharged TL11 engine with one inlet manifold and two exhaust manifolds

Number of Cylinders	Cylinder Crankshaft Position (degrees)
6	0, 120, 240, 360, 480, 600

o Engine Operating Conditions

engine speed 1500 rpm
fueling 0.06 grams of fuel per injection
static fuel injection timing 22 degrees btdc
turbine nozzle fully open
"infinite" inertia load

o Integrator Tolerance and Step Size

integrator tolerance ±0.1%, ±0.25% and ±0.5%
integrator step size 1, 2 and 4 degrees

Table 9.5a TL11 Engine Model - Summary of Experiment.

		Integration Stability Tolerance (%)		
		±0.1%	±0.25%	±0.5%
Integration Step Size (degrees)	1	0.98 rpm	1.19 rpm	1.23 rpm
	2	1.16 rpm	1.45 rpm	1.71 rpm
	4	1.03 rpm	1.55 rpm	2.04 rpm

Table 9.5b Execution Speed Results for TL11 Engine Model when using Various Integration Step Sizes and Stability Tolerances.

o Engine Configuration

6 cylinder turbocharged TL11 engine with one inlet manifold and two exhaust manifolds

Number of Cylinders	Cylinder Crankshaft Position (degrees)
6	0, 120, 240, 360, 480, 600

o Engine Operating Conditions

engine speed range 750 to 2250 rpm, every 250 rpm
fueling 0.06 grams of fuel per injection
static fuel injection timing 22 degrees btdc
turbine nozzle fully open
"infinite" inertia load

o Integrator Tolerance and Step Size

integrator tolerance ±0.25%
integrator step size 1, 1.5 and 2 degrees

Table 9.6a TL11 Engine Model - Summary of Experiment.

Simulated Engine Speed (rpm)	Serial Processing	Parallel Processing			
	Execution Speed (rpm)	Execution Speed (rpm)	Speed Up Factor	Computational Efficiency (%)	"Real Time" Factor
750	0.134	0.452	3.37	67.4	1/1659
800	0.168	0.548	3.26	65.2	1/1459
875	0.249	0.82	3.3	66.0	1/1067
1000	0.317	1.03	3.25	65.0	1/971
1250	0.395	1.34	3.39	68.0	1/930
1500	0.434	1.45	3.3	66.6	1/1038
1750	0.5	1.62	3.25	65.0	1/1082
2000	0.539	1.73	3.21	64.3	1/1153
2250	0.547	1.84	3.36	67.2	1/1223

i) Results for 2 Degree Step Size

Simulated Engine Speed (rpm)	Parallel Processing	
	Execution Speed (rpm)	"Real Time" Factor
750	0.607	1/1234
875	0.937	1/933
1000	1.1	1/908
1125	1.219	1/922
1250	1.34	1/933
1500	1.45	1/1034
1750	1.52	1/1152
2000	1.513	1/1321

ii) Results for 1.5 Degree Step Size

Simulated Engine Speed (rpm)	Parallel Processing	
	Execution Speed (rpm)	"Real Time" Factor
750	0.731	1/1025
800	0.984	1/813
875	1.14	1/765
1000	1.18	1/843
1250	1.21	1/1031
1500	1.19	1/1263
1750	1.18	1/1488

iii) Results for 1 Degree Step Size

Table 9.6b Execution Time Results for TL11 Engine Model, for Various Engine Speeds and Integration Step Sizes.

o Engine Configuration

6 cylinder turbocharged TL11 engine with one inlet manifold and two exhaust manifolds

Number of Cylinders	Cylinder Crankshaft Position (degrees)
6	0, 120, 240, 360, 480, 600

o Engine Operating Conditions

engine speed 750 to 2250 rpm, every 250 rpm
fueling 0.06 grams of fuel per injection
static fuel injection timing 22 degrees btdc
turbine nozzle fully open
"infinite" inertia load

o Integrator Tolerance and Step Size

integrator tolerance ±0.25%
integrator step size 2 degree

Table 9.7a TL11 Engine Model - Summary of Look Up Table Experiment.

Simulated Engine Speed (rpm)	Parallel Processing Without using Look Up Tables	Parallel Processing with Look Up Tables	
	Execution Speed (rpm)	Execution Speed (rpm)	Speed Up Factor
750	0.452	0.714	1.58
1000	1.03	1.65	1.6
1250	1.34	2.14	1.6
1500	1.45	2.32	1.6
1750	1.62	2.6	1.61
2000	1.73	2.80	1.62
2250	1.84	2.98	1.62

Table 9.7b Parallel Execution Time Results for the TL11 Engine Model - With and Without Look Up Tables.

o Engine Configuration

6 cylinder turbocharged TL11 engine with one inlet manifold and two exhaust manifolds

Number of Cylinders	Cylinder Crankshaft Position (degrees)
6	0, 120, 240, 360, 480, 600

o Engine Operating Conditions

engine speed 1500 rpm
fueling 0.06 grams of fuel per injection
static fuel injection timing 22 degrees btdc
turbine nozzle fully open
"infinite" inertia load

o Integrator Tolerance and Step Size

integrator tolerance ±0.25%
integrator step size 2 degree

o Additional Information

each experiment was carried out:

using and not using the MC68881 Floating Point Co-Processor

Table 9.8a TL11 Engine Model - Summary of Experiments, Performed using the MC68020 Based System.

Parallel Execution (rpm)	Serial Execution (rpm)	Processors	Speed Up Factor	Computational Efficiency
1.04	0.434	3	2.4	80.0

i) MC68000 based results

Parallel Execution (rpm)	Serial Execution (rpm)	Processors	Speed Up Factor	Computational Efficiency
2.86rpm	1.17rpm	3	2.44	81.4
2.75	2.7	Speed Improvement over MC68000 based system		

ii) MC68020 based results (no co-processor)

Parallel Execution (rpm)	Serial Execution (rpm)	Processors	Speed Up Factor	Computational Efficiency
6.13rpm	3.0 rpm	3	2.05	68.3
5.9	6.91	Speed Improvement over MC68000 based system		

ii) MC68020 based results (with MC68881 co-processor)

Table 9.8b TL11 Engine Model Execution Speed Results when Computed using the MC68020 Based System.

o Engine Configuration

6 cylinder turbocharged engine with one inlet manifold and two exhaust manifolds

Number of Cylinders	Cylinder Crankshaft Position (degrees)
6	0, 120, 240, 360, 480, 600

o Engine Operating Conditions

engine speed 1500 rpm
"infinite" inertia load

o Integrator Tolerance and Step Size

integrator tolerance ±0.1%
integrator step size 1 degree

Table 9.9a Summary of Engine Model Experiment, Performed using Various Computer Systems.

Computer System	Execution Speed (rpm)	"Real Time" Factor
VAX 11/750 Mini Computer	0.72	1/2083
12.5MHz MC68000 System (i)	0.98	1/1531
VAX 11/785 Super Mini	1.82	1/823
ICL 3980 Mainframe Computer	4.0	1/373
12.5MHz MC68020 System (ii)	4.3	1/348
12.5MHz MC68020 System (iii)	10.0	1/150
25MHz MC68020 System (iv)	20.0	1/75

(i) execution time using 5 MC68000 based slave processors

(ii) execution time using 3 12.5 MHz MC68020 based slave processors

(iii) estimated execution time for nine 12.5MHz MC68020 slave processors

(iv) as (iii) except assumes the use of 25MHz slave processors

Table 9.9b Execution Time Results for a Six Cylinder Turbocharged Engine Computed on Different Computer Systems.

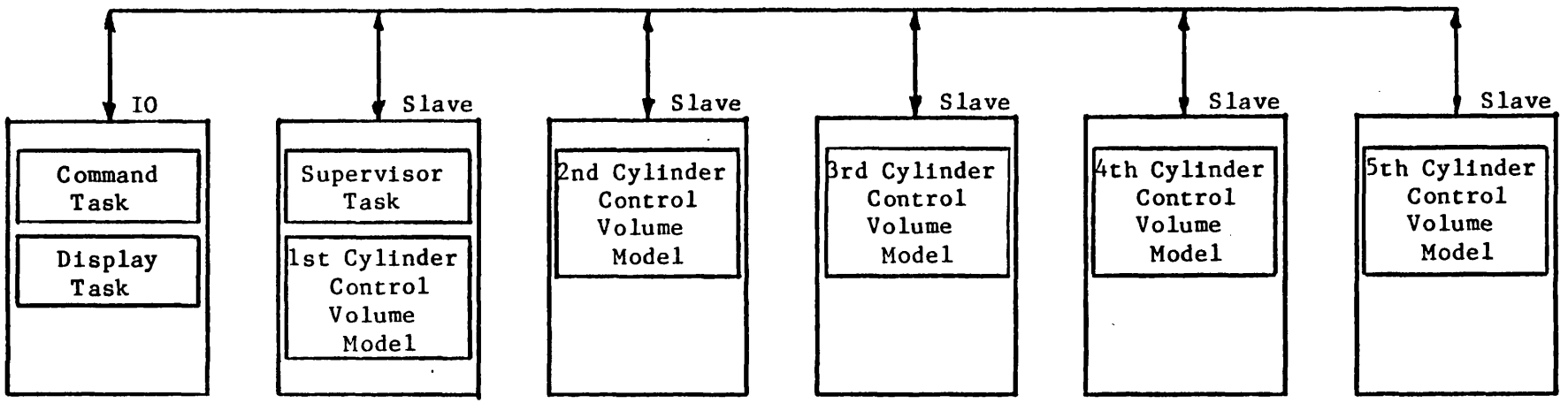


Figure 9.1 Ideal Task Allocation for Cylinder Control Volume Model Experiments.

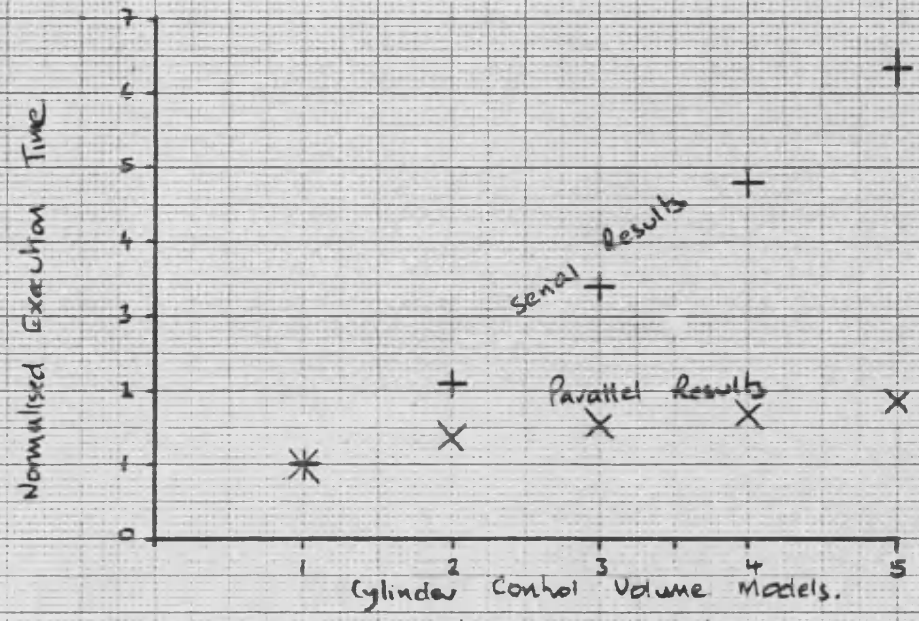


Figure 9.2a Normalised execution time, for one to five cylinder control volume models when computed serially and in parallel.

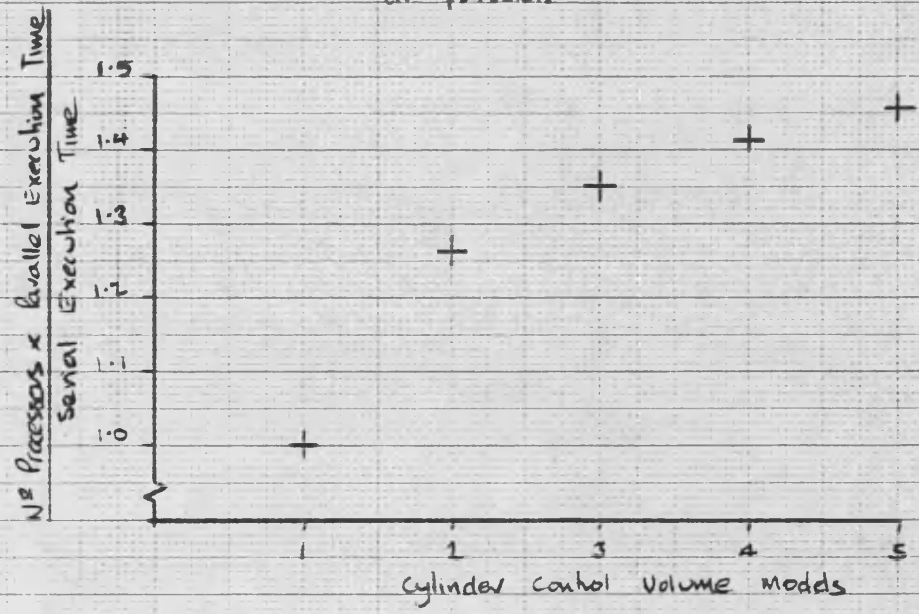


Figure 9.2b Influence of the changing computational requirements of the engine power cycle on model execution time.

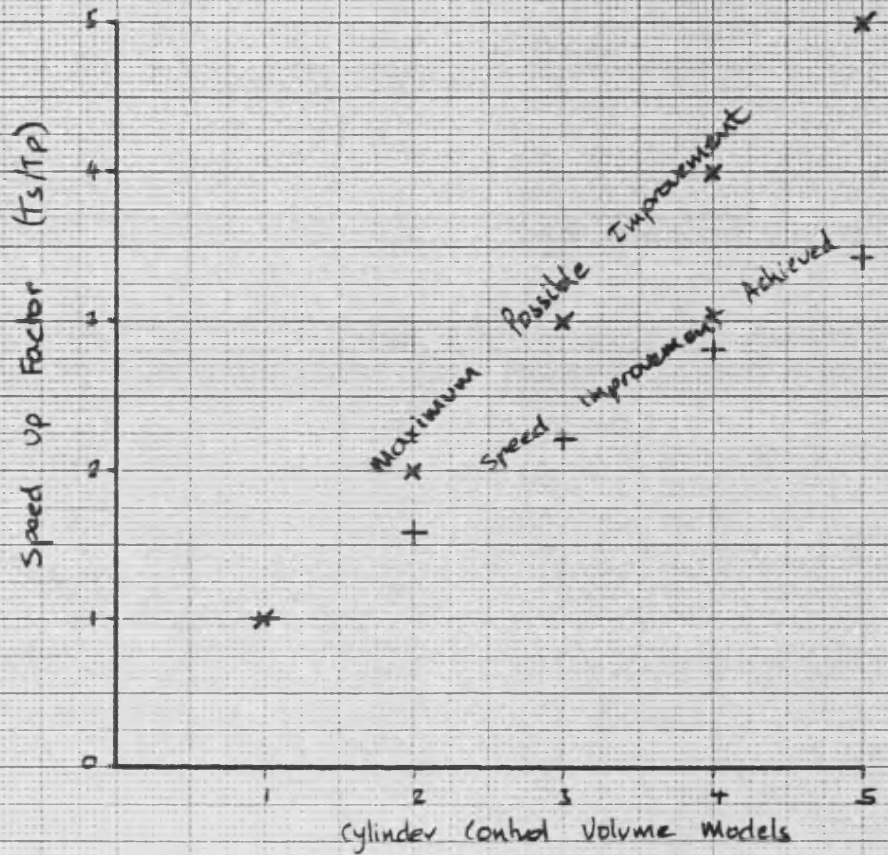


Figure 9.2c Speed up Factor for ideal task allocation.

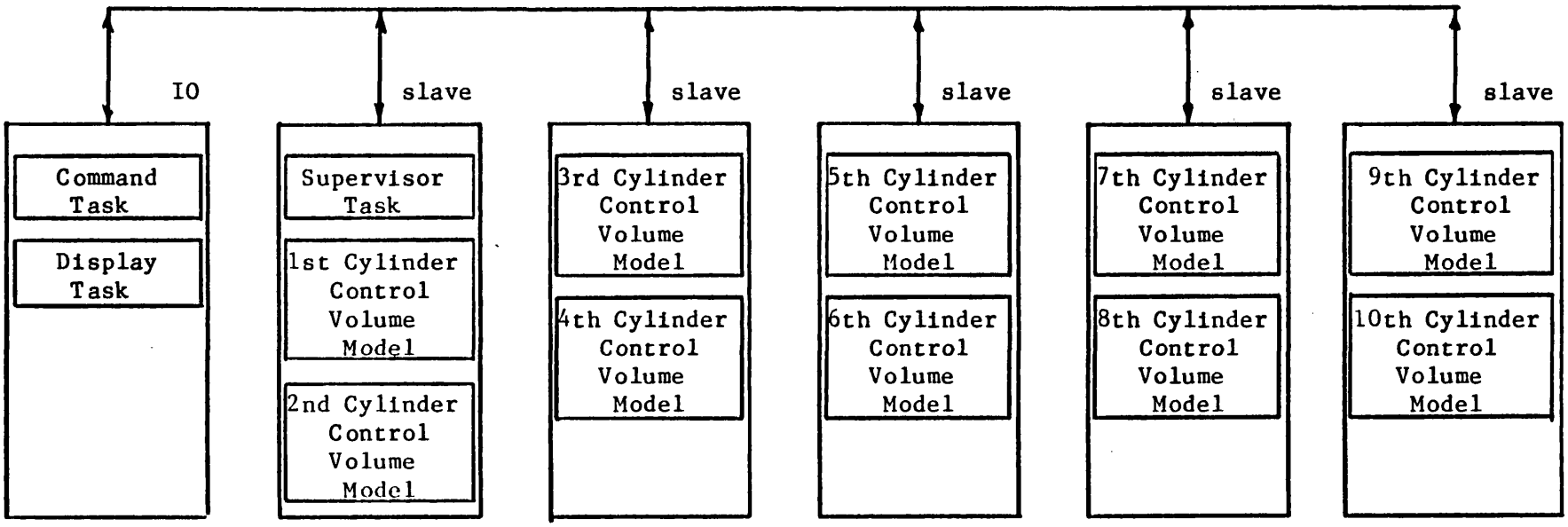


Figure 9.3 Non-Ideal Task Allocation for Cylinder Control Volume Model Experiments.

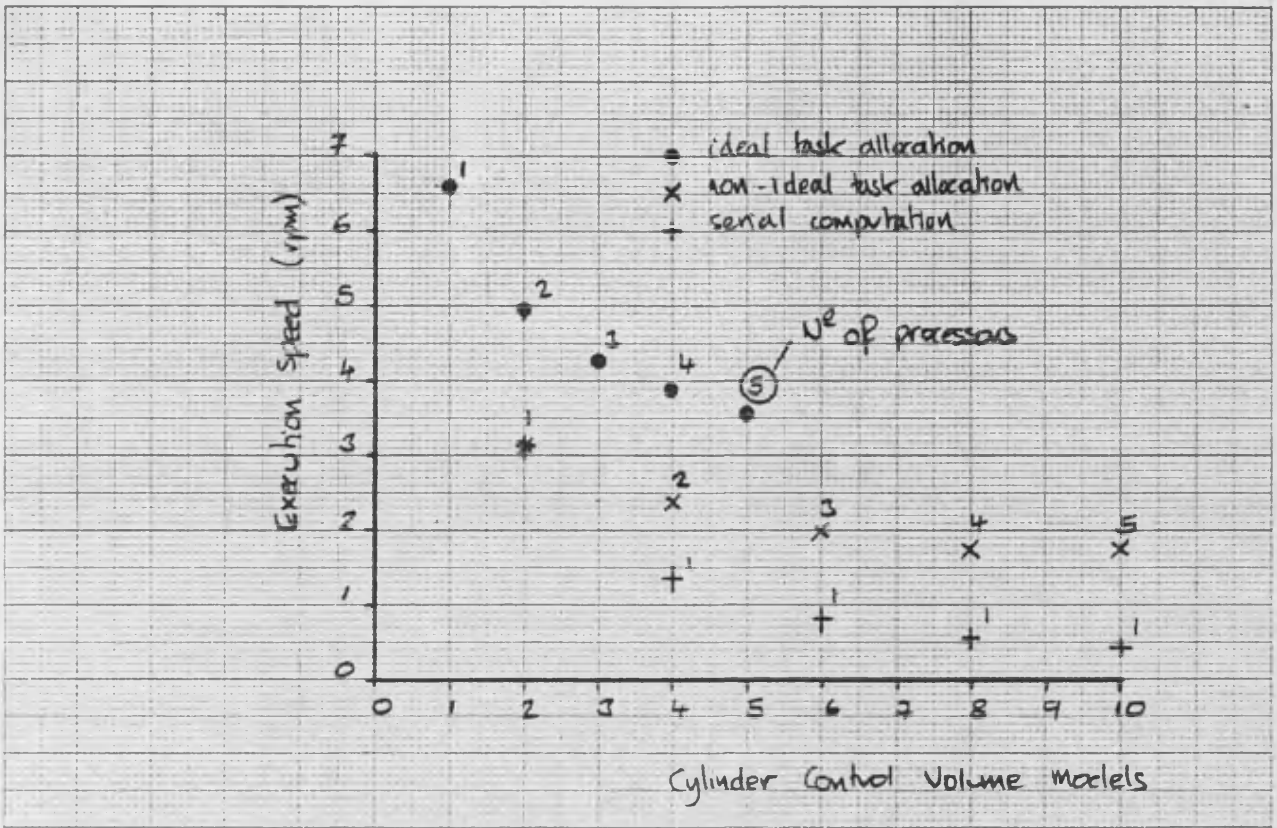


Figure 9.4a Cylinder Model - Execution time results. (ideal and non-ideal task allocations)

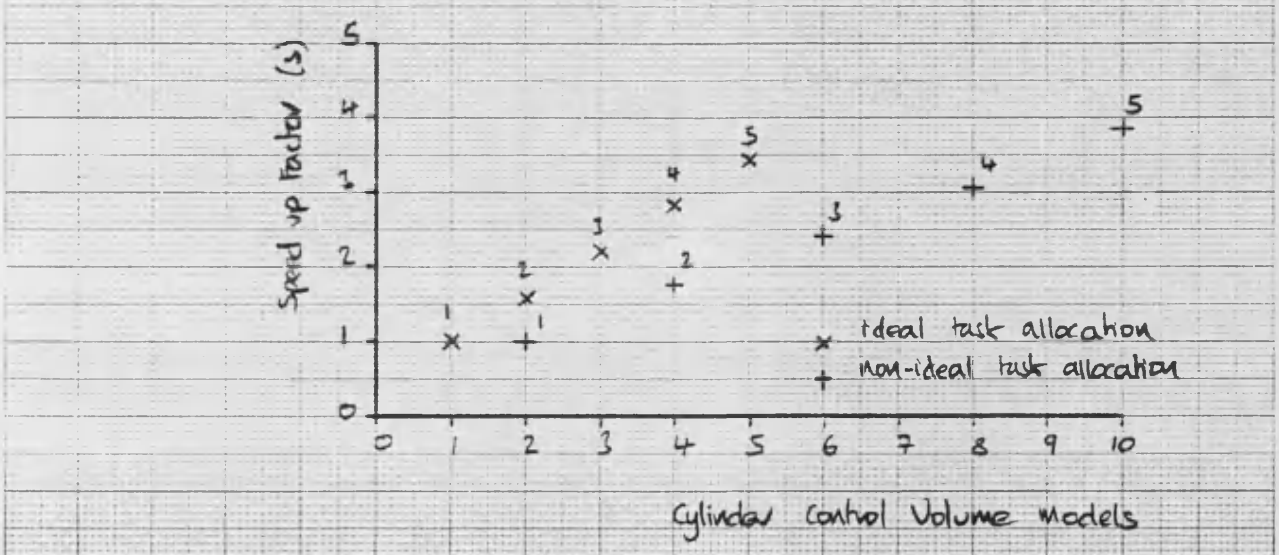


Figure 9.4b Cylinder Model - speed up Factor results (ideal and non-ideal task allocations)

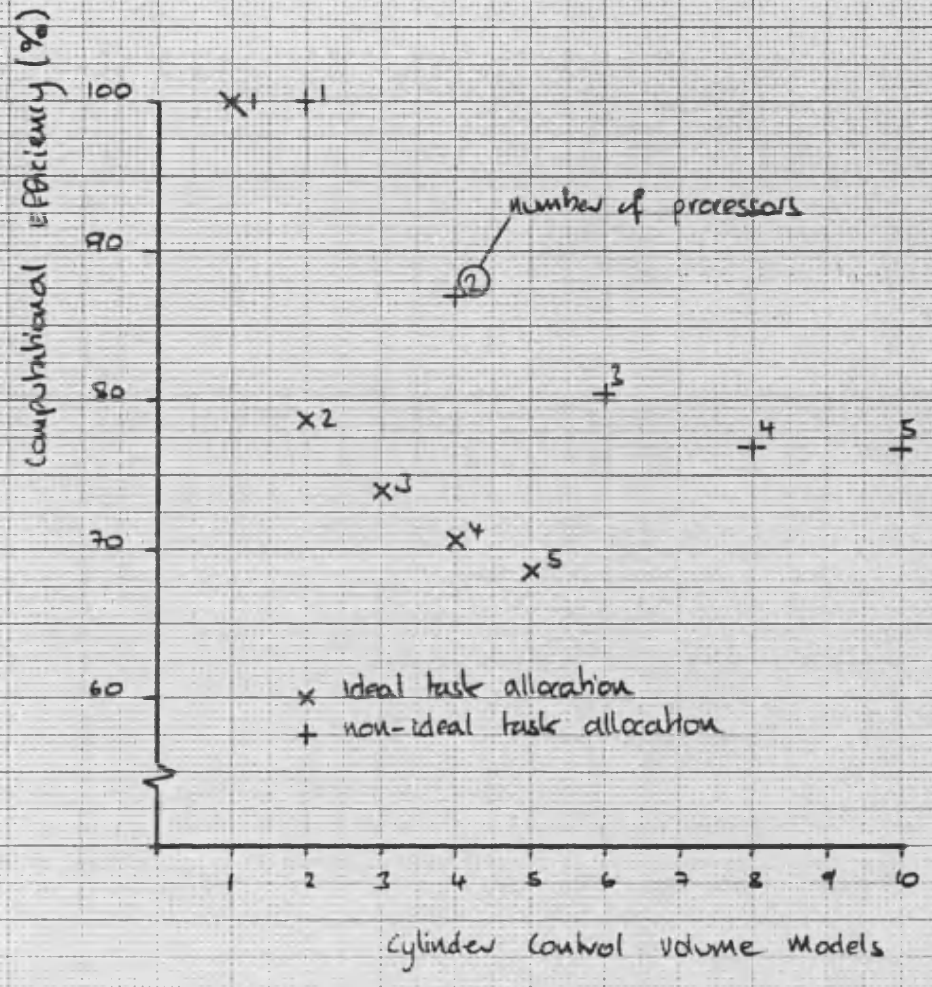
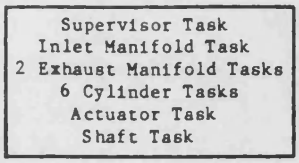
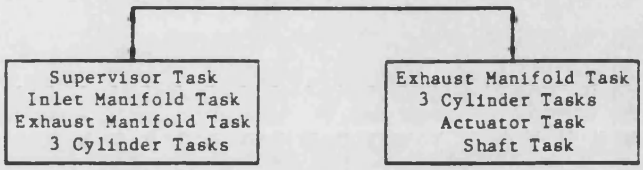


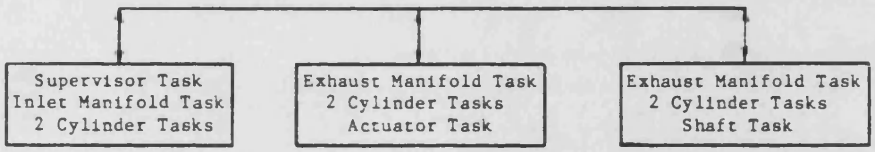
Figure 94c Computational efficiency for ideal and non-ideal task allocations.



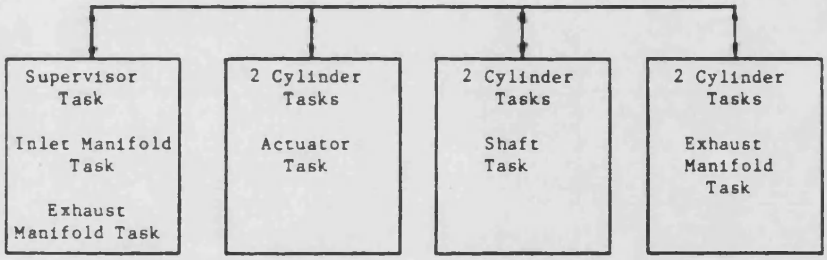
i) Single Processor Task Allocation.



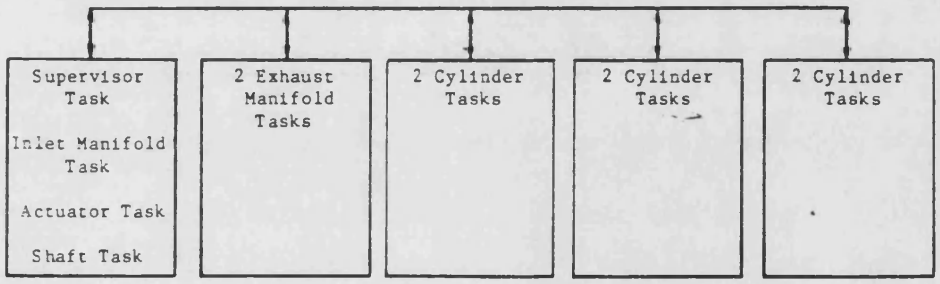
ii) Two Processor Task Allocation.



iii) Three Processor Task Allocation.



iv) Four Processor Task Allocation.



v) Five Processor Task Allocation.

During all Experiments, the IO Processor Computed the Command and Display Task.

Figure 9.5 Task Allocation for TL11 Engine Model Experiments.

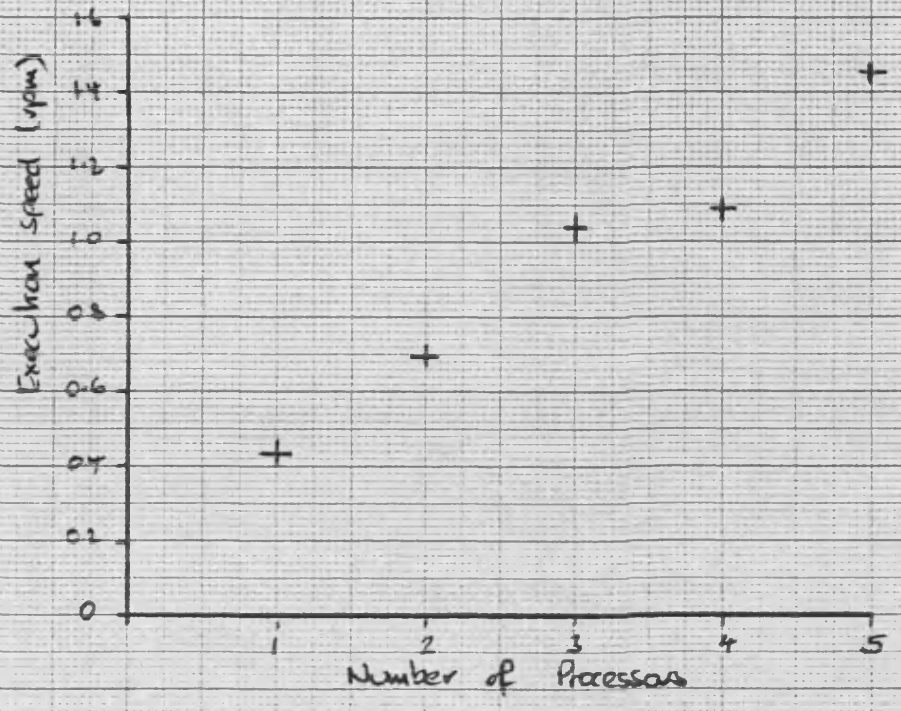


Figure 9.6 TLII engine model execution speed vs processors

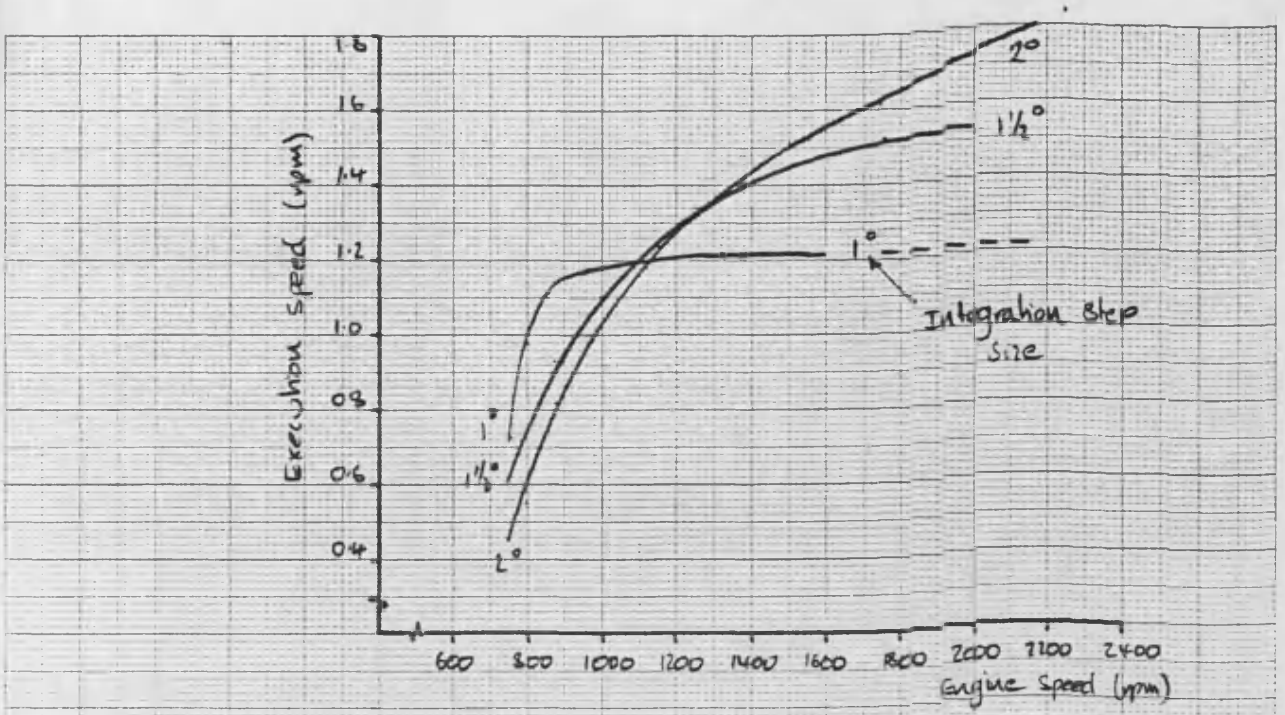


Figure 9.7a TL11 model execution speed vs engine speed

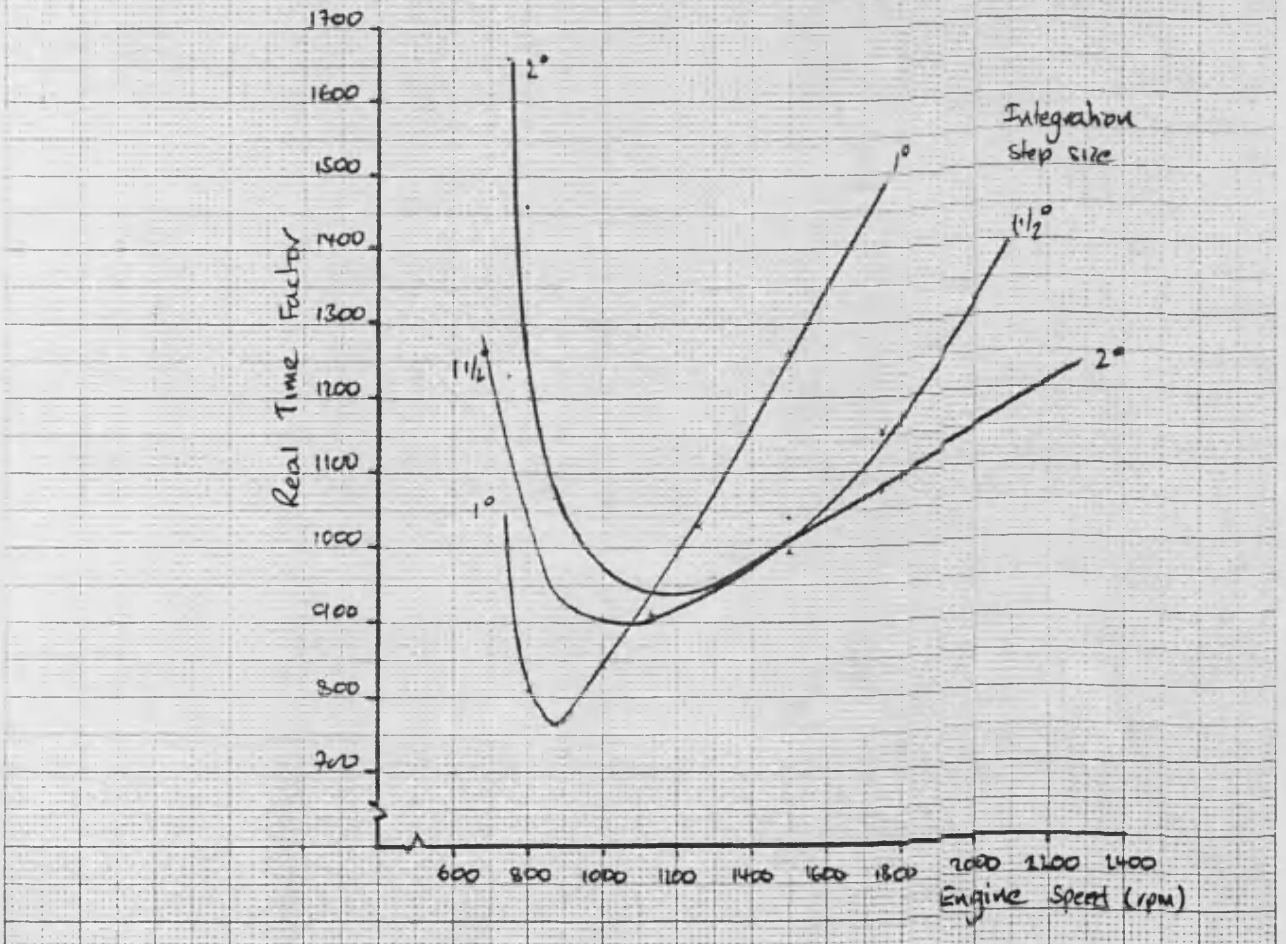


Figure 9.7b Speed up Factor vs Engine Speed

CHAPTER 10

10.1 Conclusions

Simulation of a diesel engine using a filling and emptying model has been successfully demonstrated in this work. It has been shown that the model can be readily divided into a number of modules which are suitable for computing in parallel and that this method results in a significant reduction in execution time. For example, a speed improvement of 3.3 times was achieved when computing the TL11 engine model using five MC68000 based processing nodes, and this could have been increased to about six times had an additional four processing nodes been available.

The thermodynamic control volume models are the principle computational tasks of the filling and emptying model and division of the model at this level has a number of important advantages. Only two fundamental computational tasks are required, one representing a manifold and the other a cylinder; in addition the control volume models are relatively independent of one another, which minimises the need for communication between processors. A particularly important practical advantage of the division is that, because cylinders and manifolds can be considered as the basic elements from which real engines are constructed, the same approach in simulation will apply irrespective of the actual engine configuration being studied. Consequently this will involve little change to the model software. This inherent flexibility is a distinct advantage in real engine modelling applications.

The improvement in speed which can be achieved by computing the control volume models in parallel depends upon a number of factors. The most important factor is the number of processors used to compute the model. However, it has been shown that the improvement in speed is not in direct proportion to the number of processors, and that as the number is increased, breakpoints arise at which optimum speed improvements occur. The existence of these breakpoints should be considered when planning to compute an engine model in parallel. Some factors, such as integration step size influence the execution speed of the model irrespective of whether it is computed serially or in parallel. The integration step size which results in the fastest execution speed, changes quite significantly with simulated engine speed and with the stability criteria used in the numerical integration method. Use of look-up tables also improves execution speed, for example when used with the MC68000 system, the speed increased by 1.6 times.

Obviously, the use of a more powerful processor improves execution speed, as does provision of hardware which can perform floating point calculations. Use of the MC68020 processor resulted in a speed which was three times faster than that achieved with the MC68000 processor system and when the MC68881 floating point co-processor was also used, the speed improvement doubled again.

The speed improvement possible using parallel processing is affected by the changing computational requirements of the cylinder control volume models during the different phases of the engine power cycle. This makes it impossible to achieve an exact balance in the calculations to be performed by the processors, and as more

cylinders are simulated the computational efficiency reduces. Nevertheless, the computational efficiency is not expected to fall below 50%.

The choice of a tightly coupled parallel computer system, using a single shared communication bus has proved very satisfactory in this work. In particular, it has been very easy to add more processing nodes to the system when additional software tasks have had to be included. This flexibility, combined with an operating system which allows tasks to be allocated to any processor without modification, has resulted in a very adaptable computer system and was crucial to the exploitation of the flexibility inherent in the division of the engine model itself.

The choice of BCPL for coding the model has proved to be satisfactory in that the software development time was much reduced compared to what it would have been using assembler and BCPL was also a great deal more versatile than is FORTRAN, (in which other engine models appear to have been coded). However, because of the limited support provided by BCPL for basic types such as arrays and records, and the absence of any type checking whatsoever, the software development time was undoubtedly longer than would have been achieved had a formal language such as C [10.1] or PASCAL [10.2] been used.

A diesel engine simulator which provides many advanced features has been developed, and should be of real practical value in many areas of engine simulation work. Operation of the simulator has been made as near as conveniently possible to the operation of a real engine. The operator is able to change engine controls at any

time and can see the consequences of the change displayed on the graphics system. By dedicating a single processor to the user interface, it has been possible to provide advanced facilities such as the animated graphics display without any significant effect on the speed at which the model is calculated.

It will be appreciated that the significant improvement in the execution speed of the filling and emptying model has been achieved without making any simplifying changes to the model. Even accepting that some further improvement in speed is possible, it is clear that using currently available 32-bit microprocessors, the execution speed of the model will be, at best, of the order of a few tens of times slower than real time. Nevertheless, this is many times faster than when computing the models using a conventional computer system. Use of the next generation of microprocessors can be expected to result in a further significant improvement in execution speed - prehaps to the stage where real time processing of the slower running diesels becomes a practical reality. In any event the parallel computer system consisting as it does of a relatively few low cost general purpose microprocessors provides a compact, cost effective and fast simulator for the study of engine behaviour.

10.2 References

10.1 B. W. Kernighan, D. M. Ritchie.
The C Programming Language.
1978, Prentice Hall, London.

10.2 ANSI [1981] DP 7185.1.
Specification for Computer Programming Language
Pascal Second Draft.
Dec 1981, Pascal News p1-83.

CHAPTER 11

11.1 Recommendations for Future Work

Although a substantial improvement in the execution speed of a filling and emptying engine model has been demonstrated in this work, real time solution has not yet been achieved, and should remain an important objective for future work. Further speed improvement should be possible by exploiting concurrency on a larger scale, using faster computer system hardware, improving the task scheduling and possibly using different numerical methods. However, it is strongly recommended that the flexibility of the simulator should not be compromised by the desire to improve speed. This means that the basic division of the engine model into a number of "standard" modules (e.g cylinders, manifolds, shafts etc) which can be used to represent most engine configurations should be retained. In addition, the flexibility of the computer system should be retained in order to readily accommodate additional processing nodes, and simple re-organisation of software tasks.

11.2 Exploitation of Concurrency

Although exploitation of concurrency along control volume boundaries has made a significant improvement to execution speed, the improvement soon reaches a maximum, which is a function of the number of control volumes necessary to represent the engine. However, further improvement in speed can be achieved by exploiting

concurrency within the individual control volume models. Solution of a control volume model involves computing a number of sub-models and the evaluation and integration of three state equations (4.2), (4.3), (4.4), and these calculations can be performed in parallel. For example, to compute a cylinder operating in scavenge (corrector phase of solution), one processor can calculate the flow of gas through the inlet valve, another the flow through the exhaust valve and a third heat transfer. Each processor would then evaluate and integrate a state equation involving some exchange of information and finally all three processors would participate in the calculation of the gas properties and pressure. When a cylinder is operating in compression, there are no valve flows to calculate and two of the state equations are zero; consequently, one processor should be able to perform the compression stage calculations in a similar time to the time taken by three processors performing the scavenge calculations. Similarly, during the induction and exhaust phases there is only one valve flow to be calculated and two processors should prove adequate. Thus by carefully matching the number of processors to the calculations, it should be possible to improve execution speed (possibly doubling the speed compared to when one processor is allocated to each control volume model) and possibly computational efficiency at the same time.

An assessment of the value of this approach to improving execution speed can be made most simply using a model of a single cylinder. This has the advantage that it requires only four processors, one to perform IO and the other three to compute the model.

11.3 Task Scheduling

Computing the engine model using the approach described above, may require a totally different approach to the method of task scheduling used in this research. The task scheduling adopted should permit dynamic task allocation, ie allocating the cylinder control volume tasks (‡) to processors for computation according to their phase of engine operation. Although this will incur a scheduling overhead and additional inter-processor communication, it is now thought that this will not significantly reduce execution speed.

If, in practice, this is found not to be the case, the full improvement in execution speed can still be realized using a static task allocation but, of course, additional processing nodes will be required.

11.4 Processors

Little purpose is seen in using a smaller sub-division of the engine model than that described in Section 11.2, as a means of improving execution speed. This is because the improvement in speed is governed by a law of diminishing return, - even in the ideal case doubling the execution speed requires twice as many processors.

The only viable alternative hardware solution for increasing speed is to use a more powerful processor and in this respect

---

(‡) manifold control volume models can continue to be allocated statically since their calculations do not change.

further improvements in speed are clearly dependent upon the advances made by integrated circuit manufacturers. In practice, all that can be done is to keep up to date with manufacturers' products and before choosing a computer system, to critically assess all candidate processors. This study should, of course, consider factors such as the availability of floating point support, architectural differences and their implication on the engine model solution technique, operating system support etc.

11.5 Numerical Methods and Modelling Assumptions

Modified Euler, which is the numerical integration method used in the engine model is a simple solution method and it would be a useful exercise to assess its effectiveness compared to other numerical integration techniques. In particular more efficient multi-step predictor corrector methods exist, such as the Adams-Moulton method [11.1] which uses information from several previous points to extrapolate the function to the next solution step. Although such methods involve extra calculation compared to modified Euler, they allow larger integration steps to be made and hence overall, may improve speed.

The effect of integration stability criteria on the overall accuracy of the engine model should also be investigated. This work has shown that the stability criterion has a significant influence on execution time, but the influence on overall accuracy of the model is unknown. Clearly, any relaxation of the stability criterion will result in an improved run time.

Another topic for investigation is to see whether execution speed can be improved without significantly degrading overall accuracy by introducing simplifications into the model calculations. For example, the gas property model is computed at every stage in the model solution, even though the gas property values change by only a small amount from one solution stage to the next. Calculation of the gas properties involves many floating point operations and if these were performed at the predictor stage only (and the values applied during the corrector stages), execution speed would be improved (possibly by some 20%), - without any significant reduction in accuracy.

Although these specific software recommendations may seem to be somewhat mundane and hardly likely to result in the more spectacular improvements in speed achieved using parallel processing or by using significantly faster processing nodes, they are believed to be well worth pursuing since their combined effect on speed should be significant, and achievable without a great deal of effort. However, prudence should be exercised, since except in special circumstances changes cannot be justified which reduce the usefulness of the model.

11.6 General Simulator Improvements

Although a simulator with many advanced facilities has been developed in this work, there is still scope for improvement. In particular, additional work is necessary to improve the usefulness of the simulator in many "every day" engine modelling applications.

The principle weakness of the simulator, lies in the need for the operator to have a detailed knowledge of its internal workings in order to change the model being simulated. Clearly, the general usefulness of the simulator would be greatly enhanced if it were possible to change to a different engine simply by providing a description of the engine configuration and its physical characteristics. Achieving this will require the design and development of an "intelligent" pre-processor for the simulator which can determine from the description of the engine which simulator modules are required, how they are to be linked together and how to allocate the modules for computation. The design and development of such a facility would be a major undertaking but is considered essential if the simulator is to gain widespread acceptance in general engine modelling applications.

11.7 References

11.1 C. F. Gerald.
Applied Numerical Analysis Second Edition.
1978, Addison-Wesley Publishing Company.

APPENDIX A1

This appendix lists those physical parameters which are required to model the experimental TL11 engine, using the filling and emptying model.

Engine Leyland TL11 four stroke turbocharged diesel engine

Rating: maximum power 190 kW at 2100 rpm

No of cylinders: 6

Cylinder firing order: 1 5 3 6 2 4

Cylinder bore:	127.08 mm
Stroke:	146.05 mm
Connecting rod length:	266.7 mm
Compression ratio:	15.75 : 1

effective inertia of engine: (including flywheel) 2.95 kgm^2

For each cylinder and piston:

surface area of cylinder head:	97.95 cm^2
surface area of piston crown:	184.44 cm^2
surface area of cylinder bore from top of block to 1st piston ring at tdc:	107.01 cm^2
surface area of combustion chamber:	547.12 cm^2

Valves timing: relative to cylinder at tdc during open period

exhaust valve close:	14 degrees
inlet valve close:	230 "
exhaust valve open:	494 "
inlet valve open:	710 "

geometry:

		inlet valve	exhaust valve
head diameter	(mm)	55	46.5
seat angle	(degrees)	30	30
seat width	(mm)	3.6	2.8
number/cylinder		1	1
lift diagram		see figure A1.1	

Manifolds One inlet manifold and two exhaust manifolds.

Volume of inlet manifold: 4.88 l

	front	rear
volume of exhaust manifold (l)	1.38	1.2
internal surface wall area (cm^2)	817	681

Turbocharger

Experimental variable geometry unit (variable geometry on turbine section)

compressor: H2C 8625N Z31U3
turbine: experimental mk II(b) unit

turbocharger inertia: 0.003 kgm^2
(estimate)

Fuel system

Direct injection system with variable injection timing.

CAV in line fuel pump type: Majormec P5476
Standard fuel injection timing: 22 degrees btdc
Length of fuel delivery pipes: 0.72 m
(all equal)

Load system

Hydrostatic dynamometer load system, which can be set up to provide three torque speed characteristics:

- constant torque
- constant speed
- windage

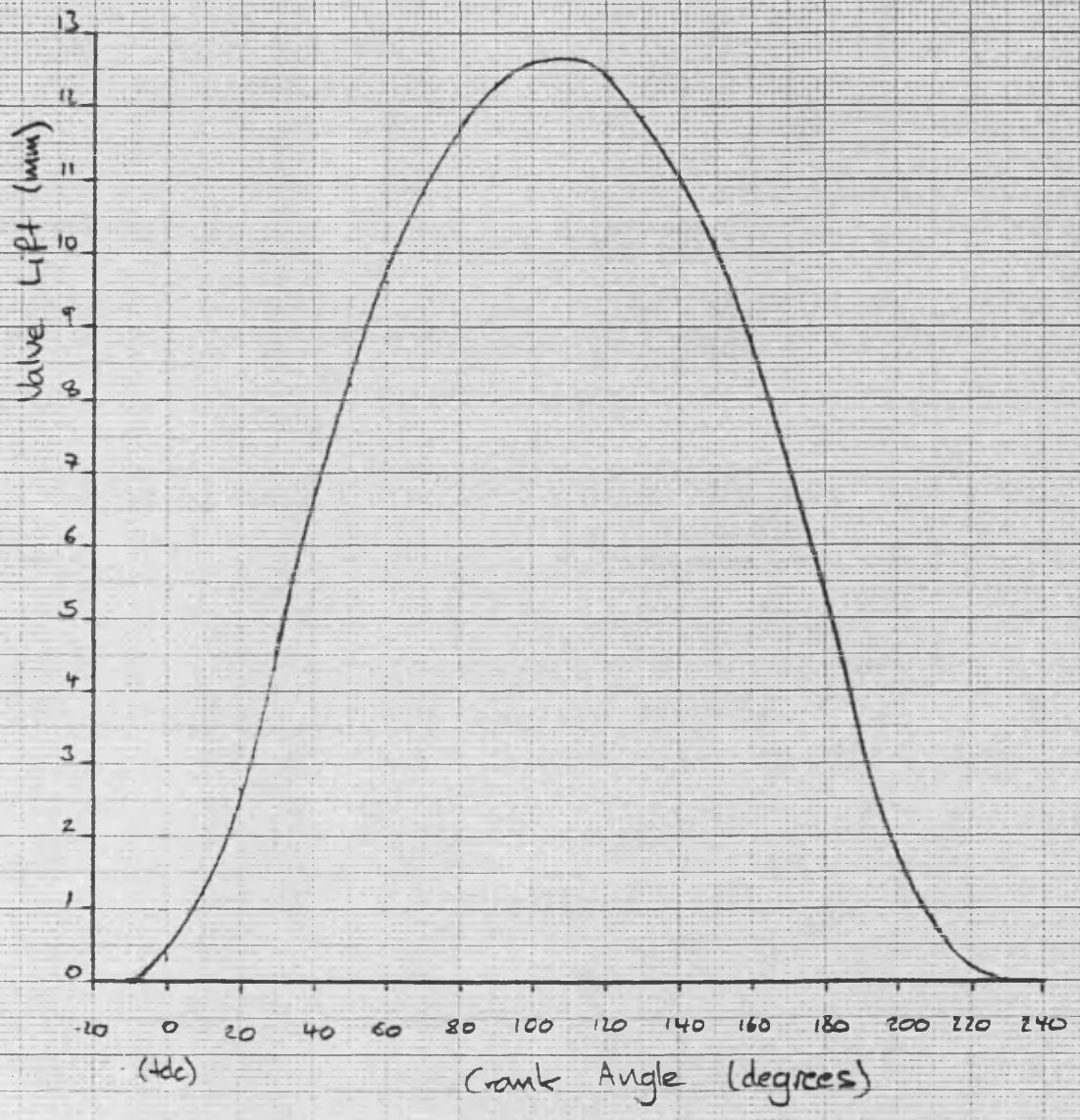


Figure A1.1 Valve lift diagram for Inlet Valve (Exhaust valve identical except crank angle offset by −216 degrees)

APPENDIX A2

A2.1 Modelling the Fuel Rack and Turbine Nozzle Control Actuators

This appendix describes the experimental programme of work which was carried out to identify models representing the dynamic behaviour of the fuel rack and turbine nozzle actuators fitted to the TL11 engine. It was not possible to identify a model of the fuel timing actuator, because it had been removed from the engine for repair.

The actuator responses were measured to step and pseudo random binary sequence (prbs) disturbance signals. The responses to the step test signals were used to assess the extent to which the actuators are slew rate limited. The responses to the prbs test signals were analysed using system identification techniques to obtain models which relate the output of each actuator to its input. The two system identification techniques used were cross-correlation [A2.1] and recursive least squares [A2.2,A2.3,A2.4,A2.5]. Having identified linear models to represent the behaviour of the actuators, velocity and saturation limits were then imposed, so that the models could then represent the response of the actuators to larger changes in position.

A2.2 Description of the Actuators, Test Equipment and Software

The fuel rack and turbine nozzle controls are actuated by the hydraulic system as shown in Figure A2.1. The flow of hydraulic

fluid to the actuator is controlled by the position of the electro-hydraulic servo valve, which is itself driven by a closed loop controller.

A digital computer was used to generate the step and prbs test signals and also to record the actuator responses. Two programs were written to generate the test signals and to record the time response of the actuators; one program generated a step wave signal, and the other generated a prbs test signal. Programs were also written to calculate the cross-correlation function between the actuator test signal and actuator response, and using recursive least squares to obtain a z-transform model of the actuator. These programs were coded using the BCPL language, but special routines, such as those required to service the signal conversion card, were written using an assembler.

A2.3 The Response of the Actuators to Step Disturbance Signals

This section presents results showing how the fuel rack and turbine nozzle actuators respond to a step change in their demanded position. The experiments were carried out to investigate to what extent the actuators are velocity limited and to obtain information about their response, (such as the settling time) which is useful to know when planning prbs experiments.

- Response of the fuel rack actuator to a step change in position.

  The response of the fuel rack actuator was measured to two step changes in position. The first test signal changed the

demanded fuel rack position from 10.5mm to 12.7mm, and the second from 10.5mm to 14.9mm. The responses of the fuel rack actuator are shown in Figure A2.2. These show that the fuel rack actuator is velocity limited (at 100mm per second); this is probably due to saturation of the flow of hydraulic fluid in the electro-hydraulic servo valve (see Figure A2.1).

- **Response of the turbine nozzle actuator to a step change in position.**

  The response of the turbine nozzle actuator was measured to a step change in its demanded turbine restriction, from 5% to 45%, - which is almost the full stroke of the actuator. The results, which are shown in Figure A2.3, indicate that the actuator has a very rapid response, achieving steady conditions approximately 50ms after the step change in demanded position. The figure also shows that the turbine nozzle actuator has a much higher velocity limit (800mm/second), than that of the fuel rack actuator (100mm/second).

A2.4 PRBS Test Programme

The response of the actuators to the prbs test signals were analysed using the recursive least squares technique to obtain a linear z-transform model, which has the form:

$$\frac{x_o(z)}{x_i(z)} = \frac{z^{-k} \sum_{i=1}^{n} b_i . z^{-i}}{1 + \sum_{i=1}^{n} a_i . z^{-i}} \qquad \text{(A2.1)}$$

where z is the shift operator

k is the time delay

n is the system order

ai and bi are the model parameters

From physical considerations, it was postulated that the actuators could be adequately represented by a second order model, with no time delay:

$$\frac{x_o(z)}{x_i(z)} = \frac{b_1 . z^{-1} + b_2 . z^{-2}}{1 + a_1 . z^{-1} + a_2 . z^{-2}} \qquad \text{(A2.2)}$$

The actuator responses were also analysed by cross-correlation to obtain the pulse responses of the actuators. Provided that noise is not correlated with the actuator input, the cross-correlation method (unlike the least squares method) results in a pulse response which is unbiased. Consequently, by comparing the pulse response obtained by cross-correlation, with the pulse response of the z-transform model any bias in the z-transform model can be detected.

Two types of prbs signal were considered for the experiments, the maximum length sequence (mls) and the inverse repeat maximum length sequence (irmls) [A2.6]. Experiments conducted using a irmls test signal take twice as long to perform as those performed using a mls signal and, of course, twice as much data has to be recorded and analysed. For these reasons the irmls is less commonly used in

system identification, although if the system responses are to be analysed by cross-correlation, its performance is superior. For example, when using the irmls, the pulse response of the system, obtained by cross-correlation, requires no correction for "dc offset". Much more importantly though, if the system tested is non-linear then its pulse response is identified more accurately, and this is particularly so if the non-linearity is directionally dependent [A2.6]. Both types of sequence were used in the test programme although all the models were obtained by analysing the responses recorded during irmls experiments.

The three signal parameters (clock period, sequence length and amplitude) were chosen according to the normal criteria and the values used are listed in Table A2.1.

During the testing the responses of the actuators were measured to 10 complete sequences of the test signal. Figure A2.4 shows a typical response of the fuel rack actuator to a mls disturbance signal, and by careful examination it will be seen that it consists of the response to 10 separate sequences. With the cross-correlation analysis, the response of the actuator to the first test sequence was disregarded because it includes the transient response to the test signal; the pulse responses obtained from the subsequent sequences were averaged in order to reduce the effect of noise. The following z-transform models were obtained:

o fuel rack actuator

$$\frac{x_o(z)}{x_i(z)} = \frac{0.004.z^{-1} + 0.0941.z^{-2}}{1 - 1.5829.z^{-1} + 0.6824.z^{-2}} \qquad \text{(A2.3)}$$

o turbine nozzle actuator

$$\frac{x_o(z)}{x_i(z)} = \frac{0.022.z^{-1} + 0.1983.z^{-2}}{1 - 1.2123.z^{-1} + 0.42249.z^{-2}} \qquad (A2.4)$$

The pulse response of the fuel rack actuator model is shown in Figure A2.5 and the pulse response of the turbine nozzle actuator model in Figure A2.7. The pulse responses obtained by cross-correlation are also shown for comparison and it will be seen that they agree closely with the pulse responses of the z-transform models. The frequency response of each actuator was obtained by calculating the Fourier transform of the pulse response, and these are shown in Figures A2.6 and A2.8. Figures A2.6a and A2.8a, show the gain response of the fuel rack and turbine nozzle actuators respectively. Similarly, Figures A2.6b and A2.8b show the phase response of the actuators. The gain and phase responses of the cross-correlation and z-transform pulse responses, agree closely to a frequency which is much higher than the natural frequency of the actuators. This close agreement gives a high degree of confidence that the z-transform models accurately represent the behaviour of the actuators, at least to small changes in position about the tested operating condition.

A2.5 Tables

Signal Parameter	Fuel Rack Test Signal	Turbine Nozzle Test Signal
Signal Type	irmls	irmls
Periodic Sequence Length	254	254
Amplitude	5v ± 0.075v	5v ± 1.125v
	11mm ± 0.165mm	25% ± 5.625%
Clock Period	2ms	4ms
Number of Sequences Applied	10	10

Table A2.1 Fuel Rack and Turbine Nozzle PRBS Test Signal Parameters

A2.6 References

A2.1 K R Godfrey.
The Theory of the Correlation Method of Dynamic Analysis and its Application to Industrial Processes and Nuclear Power Plant.
May 1969, Measurement and Control, Vol 2.

A2.2 P C Young.
Applying Parameter Estimation to Dynamic Systems - Part 1.
Oct 1969, Control Engineer.

A2.3 P C Young.
Applying Parameter Estimation to Dynamic Systems - Part 2.
Nov 1969, Control Engineer.

A2.4 B J Williams, D W Clarke.
Plant Modelling from PRBS Experiments - Part 1.
Oct 1968, Control Engineer.

A2.5 B J Williams, D W Clarke .
Plant Modelling from PRBS Experiments - Part 2.
Nov 1968, Control Engineer.

A2.6 K R Godfrey, D J Moore.
Identification of Processes Having Direction - Dependent Responses, with Gas-turbine Engine Applications.
Automatica, vol 10 p469 - 481.

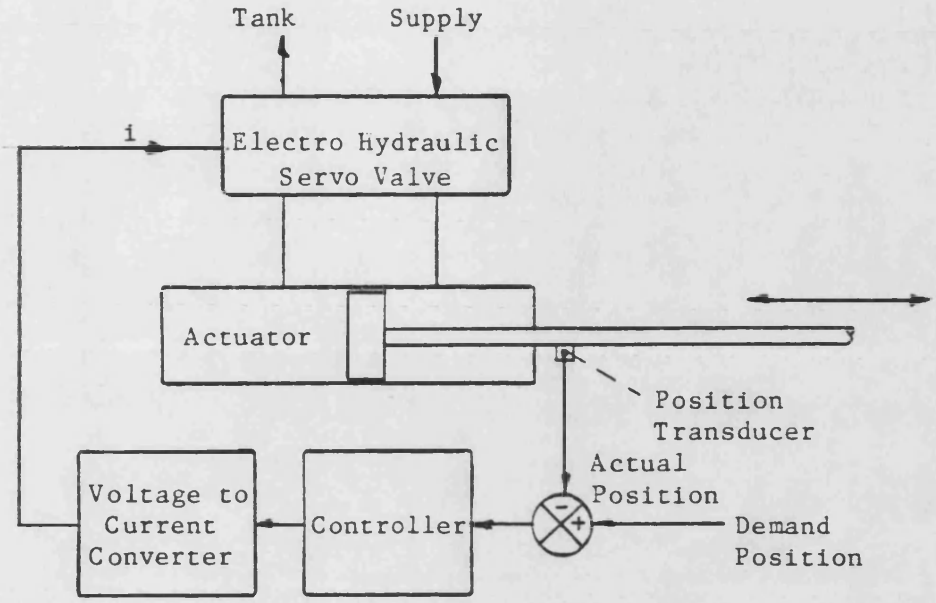


Figure A2.1 Actuation System

Fig. A2.2 Fuel rack response

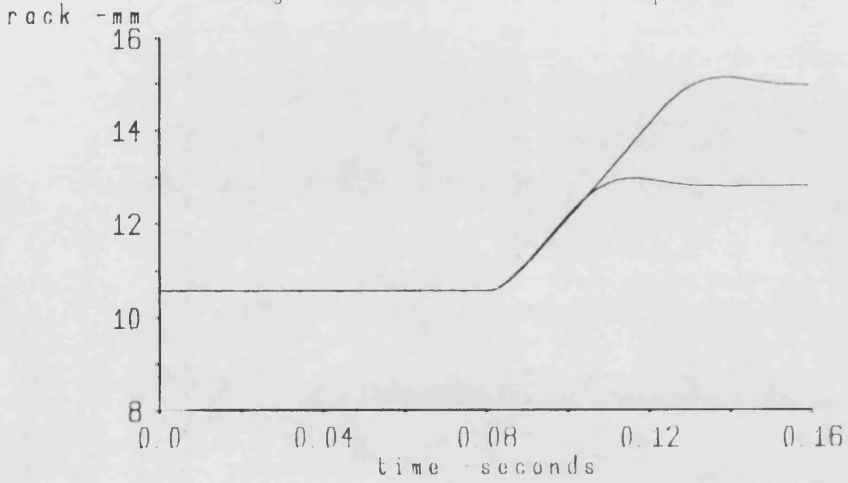


Fig. A2.3 Turbine vg response

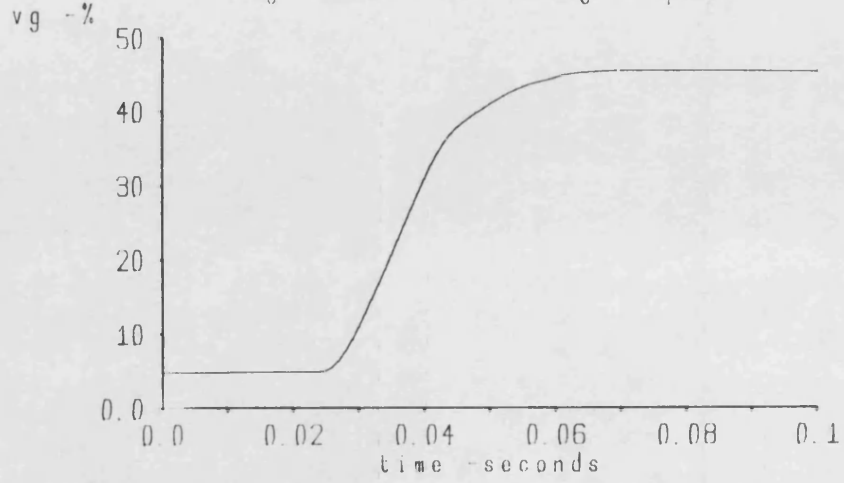


Fig. A2.4 Fuel rack response

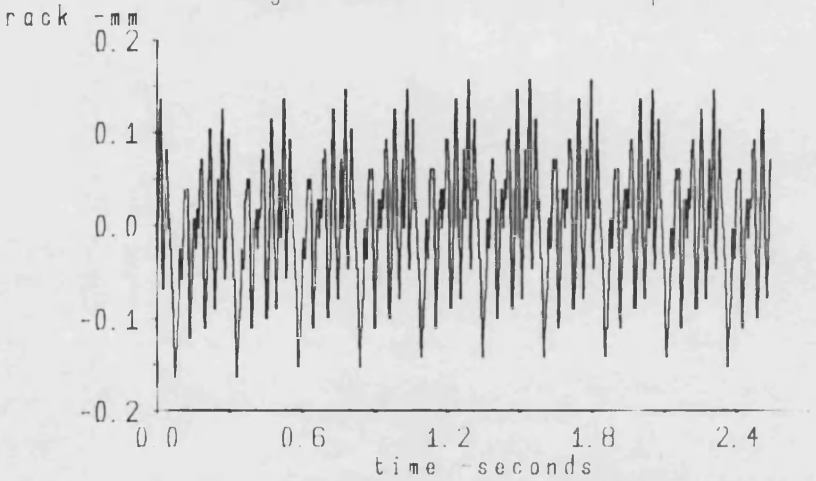


Fig. A2.5 Fuel rack pulse response

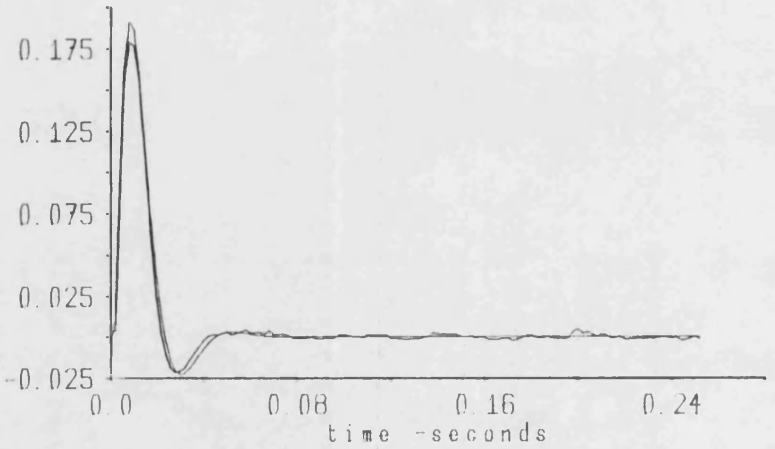


Fig. A2.6a Fuel rack gain response

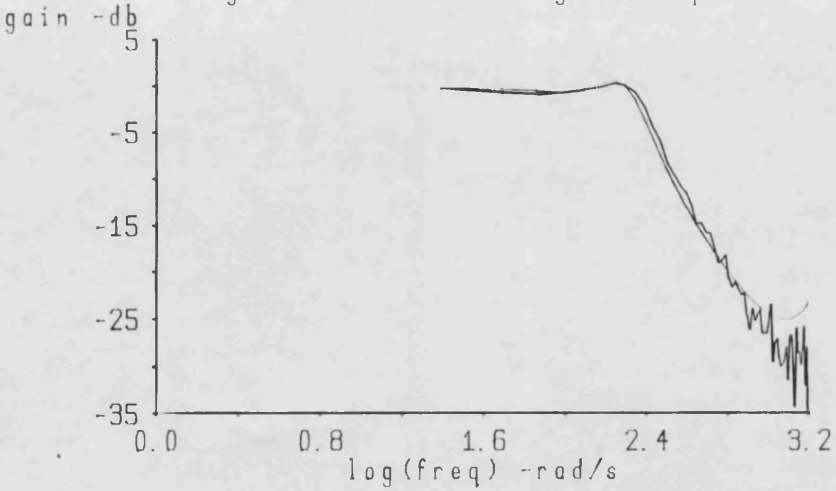


Fig. A2.6b Fuel rack phase response

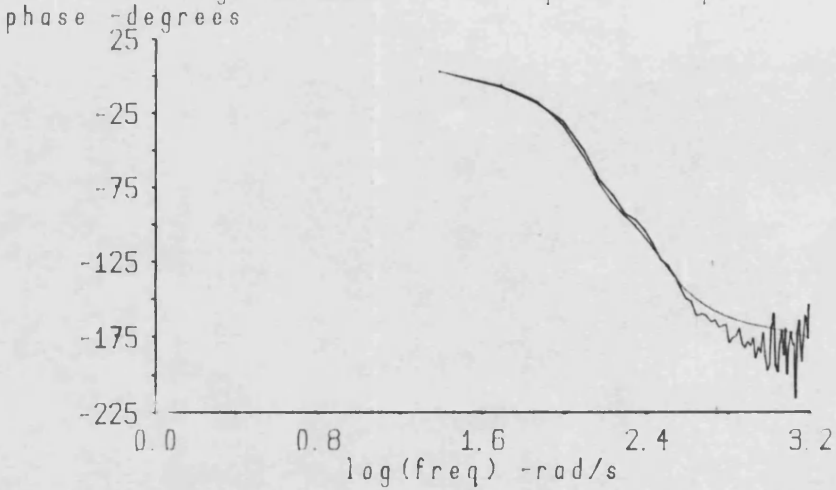


Fig. A2.7 Turbine vg pulse response

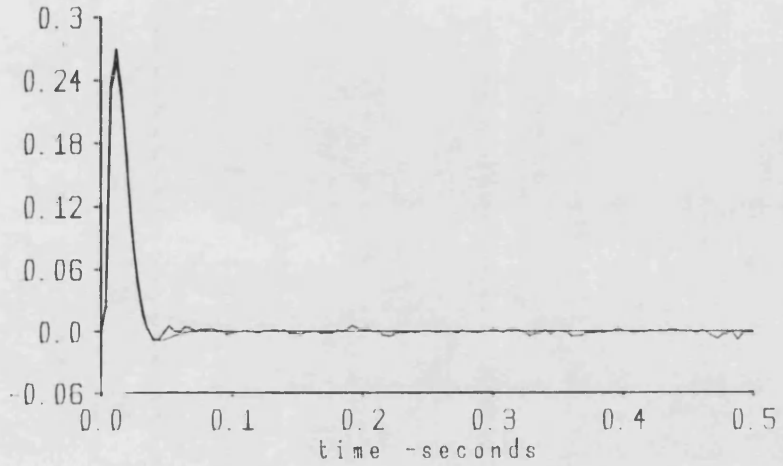


Fig. A2.8a Turbine vg gain response

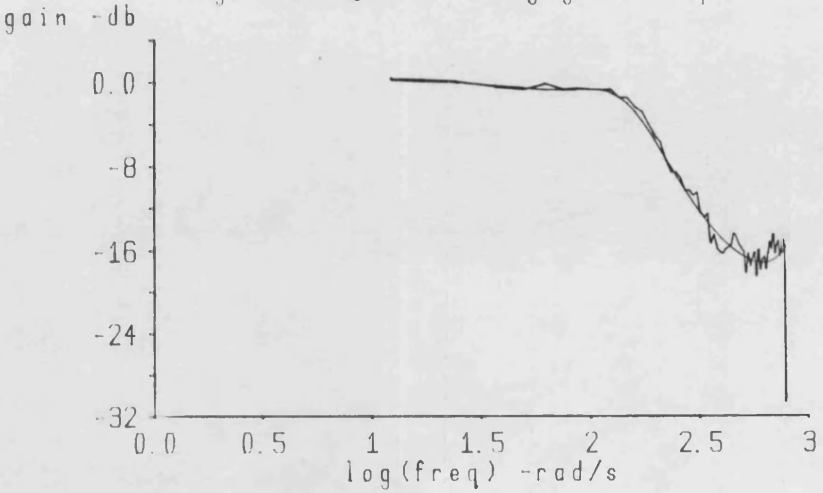
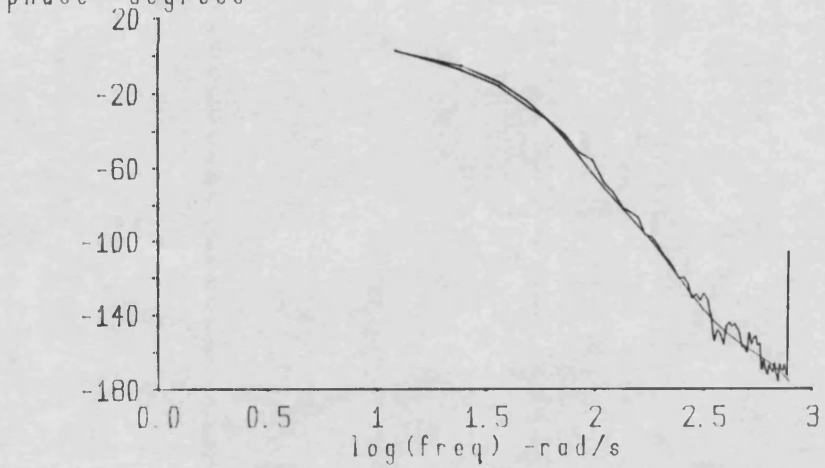


Fig. A2.8b Turbine vg phase response



APPENDIX A3

This appendix gives a description of the engine simulator commands.

o HELP — no arguments.

HELP gives a brief description of each of the simulator commands at the console.

example

```
¦help <cr>
```

o DISPLAY — On/s, off/s, cyl/k

Switches the display task on or off, or instructs the display task to obtain data from a different cylinder control volume model.

examples

```
¦display on <cr>      switches display task on
¦display off <cr>     switches display task off
¦display cyl x <cr>   use cylinder x to obtain
                      display data
```

o SEND — rack/k, inject/k, vg/k

Send the engine model new (demanded) control settings. If send is entered with no arguments then the current (demanded) control settings are listed at the console.

examples

```
¦send rack 0.006 <cr>            set the demanded rack
                                 to 6mm
¦send rack 0.006 vg 25.0 <cr>    set the demanded rack
                                 to 6mm and turbine vg
                                 to 25%
¦send <cr>                       list current (demanded)
                                 control settings
```

o STOP — no arguments

Stop the simulation and return to TRIPOS.

example

```
¦stop <cr>
```

o CYCLE — cyl/k

Display the steady state performance of a cylinder at the console

example

```
¦cycle cyl x <cr>   display steady state
                    performance data for cylinder x.
```

o CYLPLOT and CYLSAVE — cyl/k, all/s

CYLPLOT and CYLSAVE respectively plot cylinder gas responses on the monitor, or save the responses in files on disc. The variables to be plotted or saved are entered during execution of the command.

examples

```
¦cylplot cyl x <cr>   plot gas responses for
                      cylinder x
¦cylplot all <cr>     plot gas responses for all
                      cylinders
¦cylsave all <cr>     save gas responses for all
                      cylinders
```

o MANPLOT and MANSAVE — man/k, all/s

MANPLOT and MANSAVE respectively plot manifold gas responses on the monitor, or save the responses in files on disc. The variables to be plotted or saved are entered during execution of the command

example

```
¦manplot all <cr>   plot gas responses for all the
                    manifolds
```

o TIMEPLOT and TIMESAVE

We/s, tl/s, wt/s, dwt/s, fmass/s, vg/s, static/s, boost/s, T.in/s, p.exh/s, tit/s, f.exh/s, Tw/s

TIMEPLOT and TIMESAVE respectively plot engine time responses on the monitor, or save the responses to files on disc

examples

¦timeplot We <cr>	plot engine speed
¦timesave We, wt, boost <cr>	save engine speed, turbocharger speed and boost pressure

o LOGTIME

no arguments

initialise the buffer used for recording time responses.

example

¦logtime <cr>